Chapter 5

Randomization

Randomness, both as a proof technique as well as a computational resource, has a significant role in the modern theory of algorithm and complexity. In this Chapter we start our study of randomization in computation.

5.1 Basic Probability

It is perhaps one of the primal experiences, much like the primal experiences with arithmetic quantity or geometric shape, that people have come to "know" randomness. Any systematic exploration was started much later, starting with Pascal, Fermat and Laplace. But it is always a nettlesome quesiton as to what exactly is "randomness", and what exactly is "probability". I don't think this question has ever been truly satisfactorily answered, despite a lot of work on this topic. Perhaps there is no one single answer. However, brushing aside the "nature" of what is "probability", modern mathematics basically took the following perspective, after Kolmogorov. We start with an arbitrary measure space $(\Omega, \mathbf{E}, \mu)$, where Ω is some underlying set, equipped with a measure function $\mu : \mathbf{E} \to \mathbf{R}_+$, where $\mathbf{E}$ is a collection of subsets of Ω called a σ-algebra satisfying certain closure properties, and (being a probability measure) $\mu(\Omega) = 1$. (Being a measure, μ must satisfy σ-additivity, namely for disjoint countable family $\{S_i\} \subseteq \mathbf{E}$, $\mu(\bigcup_{i=1}^{\infty} S_i) = \sum_{i=1}^{\infty} \mu(S_i)$. $\mathbf{E}$ is a σ-algebra if $\Omega \in \mathbf{E}$, and $\mathbf{E}$ is closed under countable union, intersection, and complement. We will not discuss further the properties of a σ-algebra. For most of what we do, the set Ω will be finite, and $\mathbf{E}$ consists of all subsets of Ω.)

The Kolmogorov foundation of probability theory is a brilliant device to refocus the "theory of probability" as an internal mathematical subject, excluding all issues having to do with what is "randomness" as we experience it in the external world. It essentially gets rid of the issue of any reference to how this "probability theory" is to be related to the every day (or not so every day) notion of "chance" in the external world. A gambler (or any user of probability theory) has to decide what is an appropriate probability space to presume, as

a mathematical model for his particular situation at hand. Ths modeling is external to any mathematical theory of probability. The mathematics of "probability theory" in the sense of Kolmogorov only helps to "calculate" the outcomes once the basic probability of events have been assigned. Thus, in Kolmogorov's probability theory, there is no meaning of a "fair coin"; instead, one merely sets up a suitable probability space, such as $\{H, T\}$, and postulate that $\mu(H) = \mu(T) = 1/2$. It is murkier what a quantum physicist really means when he talks about the "probability" of observing this or that quantum state.

What if we want "two independent fair coin flips"? In Kolmogorov's probability theory, we can set up a product probability space, $\{H, T\}^2$, and assign $\mu(HH) = \mu(HT) = \mu(TH) = \mu(TT) = 1/4$. Note that in this 4-point space, there is, strictly speaking, no notion of time.

While the Kolmogorov foundation is fine, it does seem to be somewhat sterile, and lacking certain intuition of "probabilistic thinking". For one thing, we do like to have a primitive notion of an independent coin flip. Moreover, if after several independent fair coin flips, we decided to have another one. This should not disturb any previous probability calculation. Instead in Kolmogorov's framework we must re-constitute a new probability space all over again. Technically all previous probability calculations must be carried out now in this new probability space. Of course they take the same values, but conceptually they now take place in a different probability space. This, I find unnatural and unsatisfactory. Frequently in an algorithm we want to be able to flip more coins as we go along. And this notion of time step is natural and intuitively helpful. It is somewhat unnatural to suppose we must have a super probability space in place a priori, which in its very definition has a built-in structure of how many coin flips we can have. (There are ways around this in Kolmogorov's framework, but they all seem contrived and unnatural.)

Luckily, for almost everything we discuss here, it will be over some finite (or at most countably infinite and recursive) space. Most of the time it will just be over some $\{0, 1\}^n$, or something similar. There will be no philosophical difficulties by taking a naive approach to the concept of probability, but one which does admit a primitive notion of a new independent bit. In principle we can enumerate all the basic events and assign them "atomic probabilities" that add to one. Therefore we will take the following point of view. We will assume whenever we need we can have an independent additional coin flip (which may not be a fair coin). We will operate at a more intuitive level of "probability", as if we had a definite meaning of a physical "randomness". Thus we can say, for example, perform the following random trials independently for certain number of times with certain probability. We take this approach with the understanding that, whenever any potential difficulty should arise, we immediately retreat back to the safe cocoon of Kolmogorov foundation of measure space, and effectively say that: the only meaningful thing is what's happening in the following measure space; any implication to the "outside world" (such as what exactly does it mean by a random step) is not the responsibility of our probability analysis and any conclusions thereof.

5.1.1 Markov's Inequality

The first inequality to consider is Markov's Inequality. It deals with any random variable that take only nonnegative values and the estimate is in terms of its expectation. Technically we need to assume its has a finite expectation, however, one can apply it with abandon, for when the nonnegative random variable has infinite expectation the estimate is trivially true.

THEOREM 5.1 *Let X be a random variable such that $X \geq 0$ and $0 \leq \mathrm{E}[X] < \infty$. For all $a > 0$,*

$$\Pr[X \geq a] \quad \leq \quad \frac{\mathrm{E}[X]}{a}$$

The expectation $\mathrm{E}[X]$ is defined to be $\int X d\mu$. This theorem gives us a bound on the probability that X takes on a value that is *greater* than its expected value by a given amount. We give a quick proof of the theorem:

Proof. Let I be an indicator variable for the event $X \geq a$. That is,

$$I = \begin{cases} 1 & \text{if } X \geq a \\ 0 & \text{if } X < a \end{cases}$$

Clearly, $\mathrm{E}[I] = \Pr[X \geq a]$. Then,

$$\Pr[X \geq a] = \int I d\mu = \int_{X \geq a} 1 d\mu \leq \int_{X \geq a} \frac{X}{a} d\mu \leq \int \frac{X}{a} d\mu = \frac{\mathrm{E}[X]}{a}.$$

♣

5.1.2 Chebyshev Inequality

THEOREM 5.2 *For any random variable X with finite $\mathrm{E}[X]$ and $\mathrm{Var}(X)$, and let $a > 0$,*

$$\Pr[|X - \mathrm{E}[X]| \geq a] \leq \frac{\mathrm{Var}(X)}{a^2}$$

Here the variance $\mathrm{Var}(X)$ is defined as $\mathrm{E}[(X - \mathrm{E}[X])^2] = \mathrm{E}[X^2] - \mathrm{E}[X]^2$, and technically we must assume it is finite. However, the above estimate is (trivially) true even if it is infinite.
Proof. Define a random variable $Y = (X - \mathrm{E}[X])^2$. Then,

$$\begin{aligned} \Pr[|X - \mathrm{E}[X]| \geq a] &= \Pr[Y \geq a^2] \\ &\leq \frac{\mathrm{E}[Y]}{a^2} \\ &= \frac{\mathrm{Var}(X)}{a^2} \end{aligned}$$

In the above derivation, the inequality is obtained using the Markov's inequality. ♣

5.1.3 Chernoff Bound

It may appear that Markov's inequality is pretty weak. However it is very applicable in many situations, mainly because it only assumes that the random variable under consideration to be non-negative and does not assume anything about its distribution. Judiciously applied it can yield remarkably sharp bounds. In this section, we use it to prove the Chernoff bound, which is a powerful inequality dealing with sums of independent random variables.

Before we give the statement of the Chernoff bound, we introduce the variables it will use. Let $\{X_1, X_2, \cdots, X_n\}$ be a set of n independent, identically distributed random variables such that each X_i has

$$\Pr[X_i = 1] = \Pr[X_i = -1] = \frac{1}{2}.$$

Let S_n be their sum: $S_n = \sum_{i=1}^{n} X_i$.

THEOREM 5.3 *For all n, and for all $a > 0$,*

$$\Pr[S_n \geq a] \quad \leq \quad e^{-a^2/2n}$$

Before proving this theorem, we note that it can be restated as

$$\Pr[S_n \geq \epsilon \cdot n] \quad \leq \quad e^{-\frac{1}{2}\epsilon^2 n} = (e^{-\frac{1}{2}\epsilon^2})^n$$

or

$$\Pr[S_n \geq \alpha \cdot \sqrt{n}] \quad \leq \quad e^{-\frac{1}{2}\alpha^2}$$

If we think of ϵ and α as positive constants, then the first inequality says the probability of having an $\Theta(n)$ deviation from expectation (0) is exponentially small, and the second inequality says that this "tail probability" balances out at $\Theta(\sqrt{n})$ away from expectation. This last statement is in accord with the central limit theorem, which states that

$$\lim_{n\to\infty}[S_n \geq \alpha \cdot \sqrt{n}] = \int_{\alpha}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx.$$

The advantage of the Chernoff Bound is that it is valid for all n and a, and not merely a statement of limit.

Now, we prove the theorem.

Proof. The trick is to consider the exponentiation of S_n, namely the random variable $e^{\lambda S_n}$. Here $\lambda > 0$ is some number to be fixed later. We compute the expectation of $e^{\lambda S_n}$ as follows.

$$\mathrm{E}\left[e^{\lambda S_n}\right] = \mathrm{E}\left[e^{\lambda \sum_{i=1}^{n} X_i}\right] = \mathrm{E}\left[\prod_{i=1}^{n} e^{\lambda X_i}\right] = \prod_{i=1}^{n} \mathrm{E}\left[e^{\lambda X_i}\right]$$

To get the last equality, we use the assumption that the $\{X_i\}$ are independent random variables and hence $\{e^{\lambda X_i}\}$ are also independent. (Recall that, for independent random

variables X and Y, the expectation is multiplicative $\mathrm{E}[XY] = \mathrm{E}[X]\,\mathrm{E}[Y]$.) For any i, the random variable $e^{\lambda X_i}$ takes values e^{λ} and $e^{-\lambda}$, each with probability $1/2$. Its expectation is

$$\mathrm{E}\left[e^{\lambda X_i}\right] = \cosh(\lambda) = \frac{e^{\lambda} + e^{-\lambda}}{2}$$

The above quantity is bounded by

$$\frac{e^{\lambda} + e^{-\lambda}}{2} \leq e^{\lambda^2/2}.$$

One can prove the above inequality by analyzing the Taylor expansion of LHS and RHS. We do this analysis at the end. Using this bound, we have

$$\mathrm{E}\left[e^{\lambda S_n}\right] \leq \prod_{i=1}^{n} e^{\lambda^2/2} = e^{\lambda^2 n/2}$$

We can now apply Markov's inequality to get:

$$\Pr[S_n \geq a] = \Pr[e^{\lambda S_n} \geq e^{\lambda a}] \leq \frac{\mathrm{E}\left[e^{\lambda S_n}\right]}{e^{\lambda a}} \leq \frac{e^{\lambda^2 n/2}}{e^{\lambda a}}$$

The above inequality is true for any λ, so we are now free to choose λ to optimize it. To get the tightest upper bound, we minimize the the exponent in the above function. Standard calculus technique shows that a minimum occurs at $\lambda = a/n$. This gives us

$$\Pr[S_n \geq a] \quad \leq \quad e^{-a^2/2n}$$

Finally, we show that

$$\frac{e^{\lambda} + e^{-\lambda}}{2} \leq e^{\lambda^2/2}.$$

Taylor expansion of LHS is

$$\cosh \lambda = 1 + \frac{\lambda^2}{2!} + \frac{\lambda^4}{4!} + \cdots$$

Taylor expansion for the function e^x shows that

$$e^{\lambda^2/2} = 1 + \frac{\lambda^2}{2} + \frac{(\lambda^2/2)^2}{2!} + \frac{(\lambda^2/2)^3}{3!} + \cdots$$

Compare the two series term by term. The kth term of LHS is $\lambda^{2k}/(2k)!$, while that of $e^{\lambda^2/2}$ is $(\lambda^2/2)^k/k!$. Focusing only on the even factors, we have $(2k!) \geq 2^k \cdot k!$. Thus,

$$\cosh \lambda \leq e^{\lambda^2/2}.$$

There are several different forms of Chernoff Bound which will be useful. They all concern tail probabilities of sums of independent random variables. Let $\{X_1, X_2, \cdots, X_n\}$ be a set of n independent 0-1 random variables, with

$$\Pr[X_i = 1] = p_i, \quad \Pr[X_i = 0] = 1 - p_i,$$

and let $p = \sum_{i=1}^{n} p_i/n$. We define the centralized random variables $\{Y_1, Y_2, \cdots, Y_n\}$ where $Y_i = X_i - p_i$, then

$$\Pr[Y_i = 1 - p_i] = p_i, \quad \Pr[Y_i = -p_i] = 1 - p_i,$$

and $\mathrm{E}[Y_i] = 0$. Let $S_n = \sum_{i=1}^{n} Y_i = \sum_{i=1}^{n} X_i - pn$.

THEOREM 5.4 *For any $\Delta > 0$,*

$$\Pr[S_n \geq \Delta] \leq e^{-2\Delta^2/n}.$$

By symmetry, the bound applies to $-S_n$ as well, and so

$$\Pr[|S_n| \geq \Delta] \leq 2e^{-2\Delta^2/n}.$$

Note that when we take $\Delta = \delta pn$, then

$$\Pr[|\sum_{i=1}^{n} X_i - pn| > \delta pn] < 2e^{-2\delta^2 p^2 n}.$$

Here is another form. Let $\{X_1, X_2, \cdots, X_n\}$ be a set of n independent 0-1 random variables, with $\Pr[X_i = 1] = p$. Then

THEOREM 5.5 *For any $0 < \delta < 1/2$,*

$$\Pr[\sum_{i=1}^{n} X_i > (1+\delta)pn] < e^{-\delta^2 pn/4}$$

and

$$\Pr[\sum_{i=1}^{n} X_i < (1-\delta)pn] < e^{-\delta^2 pn/2}.$$

Theorem 5.5 is better than Theorem 5.4 for small p.

For not necessarily 0-1 random variables,

THEOREM 5.6 *Let X_i, $1 \leq i \leq n$, be mutually independent with all $\mathrm{E}[X_i] = 0$ and all $|X_i| \leq 1$. Let $S_n = \sum_{i=1}^{n} X_i$. Then for all $a > 0$,*

$$\Pr[S_n > a] < e^{-a^2/2n}.$$

The proofs of these versions of Chernoff Bound all follow similar lines.

A version of this type of bound also holds with hypergeometric distribution. Randomly pick n balls without replacement, from N black and white balls, with pN black balls. Let S be the number of black balls among n balls picked.

Then

THEOREM 5.7 *For any* $0 \leq t \leq 1$,

$$\Pr[|S/n - p| \geq t] \leq 2e^{-2nt^2}.$$

This is known as the *Hoeffding bound*.

5.1.4 Universal Hashing

DEFINITION 5.8 (UNIVERSAL FAMILY OF HASH FUNCTIONS) *Let U and T be finite sets. Let S be an index set for a family of functions $\{h_s : U \to T\}_{s \in S}$. $\{h_s\}_{s \in S}$ is called a* universal family of hash functions *if* $\forall \alpha, \beta \in T, \forall x, y \in U, x \neq y$,

$$\Pr_{s \in S}[h_s(x) = \alpha \wedge h_s(y) = \beta] = \frac{1}{|T|^2}$$

Notice that the RHS of the equation above $1/|T|^2$ is the probability of getting α and β when we choose two elements independently and uniformly at random from T.

The proper treatment in the Kolmogorov framework will be to define a measure space with the underlying set S, equipped with the uniform distribution. Then for all fixed $x \in U$, the map $Z_x : s \mapsto h_s(x)$ is a random variable.

We have $\forall x \neq y \in U$, $\forall \alpha, \beta \in T$, $\Pr_{s \in S}[Z_x(s) = \alpha \wedge Z_y(s) = \beta] = \frac{1}{|T|^2}$. Hence, $\forall \alpha \in T, \forall x \in U$, pick $y \in U$, and $y \neq x$, (we assume $|U| \geq 2$),

$$\begin{aligned}
\Pr_{s \in S}[Z_x(s) = \alpha] &= \sum_{\beta \in T} \Pr_{s \in S}[Z_x(s) = \alpha \wedge Z_y(s) = \beta] \\
&= \sum_{\beta \in T} \frac{1}{|T|^2} \\
&= \frac{1}{|T|}
\end{aligned}$$

So, Z_x is a uniform random variable on T. It follows that for any $x \neq y \in U$ and $\alpha, \beta \in T$,

$$\Pr_{s \in S}[Z_x(s) = \alpha \ \wedge \ Z_y(s) = \beta] = \Pr_{s \in S}[Z_x(s) = \alpha] \cdot \Pr_{s \in S}[Z_y(s) = \beta].$$

So, for any $x \neq y \in U$, the random variables Z_x and Z_y are independent. The definition of universal hash function is equivalent to asking a set of *pairwise independent* and unifromly distributed random variables, $\{Z_x\}_{x \in U}$. The random variables in this set may be jointly dependent, but any two of them are independent.

However, instead of thinking in terms of pair-wise independent random variables $Z_x(s)$, we rather think of $h_s(x) = Z_x(s)$ as a random map from U to T, by randomly choosing an index $s \in S$. The two views are of course completely equivalent here.

An Example: Let p be a prime number. Then $\mathbb{Z}/p = \{0, 1, \ldots, p-1\}$ with the operations $+$ and $\cdot$ is a finite field. Consider the map $h_{s=(a,b)} : x \mapsto ax + b$ for $a, b \in \mathbb{Z}/p$. We will verify that $\{h_{(a,b)}\}_{a,b\in\mathbb{Z}/p}$ is a universal family of hash functions.

For all $x, y, \alpha, \beta \in \mathbb{Z}/p, x \neq y$, how many pairs $(a, b) \in \mathbb{Z}/p$ are there satisfying the following equations?

$$\begin{aligned} ax + b &= \alpha \\ ay + b &= \beta \end{aligned}$$

(In the above equations, a and b are the unknowns.) The determinant of this 2×2 linear system is

$$\det \begin{pmatrix} x & 1 \\ y & 1 \end{pmatrix} = x - y \neq 0.$$

Therefore, there exists a unique solution such that this equation holds. Thus,

$$\Pr_{s=(a,b)\in(\mathbb{Z}/p)^2}[h_s(x) = \alpha \wedge h_s(y) = \beta] = \frac{1}{p^2}.$$

So, $\{h_{(a,b)}\}_{a,b\in\mathbb{Z}/p}$ is a universal family of hash functions.

This can be generalized to any finite field $\mathrm{GF}(p^n)$. It is known that for any prime p and any $n \geq 1$, there is a finite field of p^n elements. Up to isomorphism such a field is unique. In particular, for any $k \geq 0$, the polynomial $X^{2\cdot 3^k} + X^{3^k} + 1$ is an irreducible polynomial in $\mathbb{Z}_2[X]$, and therefore we have an explicit finite field in the form of

$$\mathbb{Z}_2[X]/(X^{2\cdot 3^k} + X^{3^k} + 1).$$

In the definition of this particular family of universal hash functions via affine linear functions $ax + b$, if it is defined over a finite field $\mathrm{GF}(2^n)$, we can truncate any number of bits from n to make $|T| = 2^k$, for any $0 \leq k \leq n$. Whenever in the following we speak of a family of universal hash functions, unless otherwise stated, we always refer to this family of affine linear functions, and if necessary, over that particular family of finite fields $\mathrm{GF}(2^n)$, with $n = 2 \cdot 3^k$.

The power of universal hash functions come from the fact that on the one hand they behave more or less like a random function, on the other hand they can be succintly specified by only $2n$ bits.

5.2 Randomized Algorithms: MAXCUT

As a taste for randomized algorithms, we discuss the graph problem MAXCUT. Let $G = (V, E)$ be an undirected graph over n vertices. A cut C of G is a partition of vertices V into disjoint union $V_1 \cup V_2$. We also identify a cut with the set of edges between V_1 and V_2, i.e., $e \in C$ consists of those edges with one of its incident vertices in V_1 and the other in V_2.

DEFINITION 5.9 (MAXCUT) *A* MAXCUT *of a graph $G = (V, E)$ is a cut C such that $|C|$ is maximized over all cuts of G.*

Similar to MAXCUT, MINCUT of G is defined as the minimum $|C|$ over all cuts of G. We know that MINCUT is in P. It can be solved using maximum network flow between all pairs of vertices.

We know that MAXCUT is NP-complete, therefore we do not expect to solve it efficiently. However, we can look for approximate solutions. To quantify the accuracy of our approximations we will introduce a new term. We want a polynomial-time algorithm that achieves a cut C such that

$$\frac{|C|}{|C^*|} \geq r$$

where C^* is a maximum cut. Such an algorithm is called an r-approximation.

5.2.1 Deterministic MAXCUT Approximation Algorithm

We can define a greedy algorithm that achieves a 1/2-approximation:
For a graph $G = (V, E)$ with $V = \{1, \ldots, n\}$, define $E_i = \{(k, i) \in E : k < i\}$. Initially, let $V_1 = \{1\}$ and $V_2 = \emptyset$. Then, for each i from 2 to n, add i to either V_1 or V_2 so that the number of edges in E_i that are on the cut is maximized, i.e., put i in V_1 iff $|\{k \in V_2 : (k, i) \in E_i\}| \geq |\{k \in V_1 : (k, i) \in E_i\}|$. We claim that this heuristic achieves 1/2-approximation.

Let C be the cut obtained by the algorithm. The disjoint sets $E_1, E_2, \ldots, E_n$ partition E. So, $|E| = \sum_i E_i$. For each $i \in V$, let $E'_i = E_i \cap C$. Then, $C = \bigcup_i E'_i$. As sets E_i are disjoint, the sets E'_i are also disjoint. Thus, $|C| = \sum_i |E'_i|$. The main observation is that for each $i \in V$, we have $E'_i \geq E_i/2$. We conclude that $|C| \geq |E|/2$. As the size of maximum cut $|C^*| \leq |E|$, $|C| \geq |C^*|/2$.

5.2.2 Randomized MAXCUT Approximation Algorithm

We present a randomized 1/2-approximation algorithm for MAXCUT. Then we show that it can be derandomized in polynomial time. This example illustrates the ideas of randomization and derandomization in a simple setting.

The randomized algorithm is very simple. Assign a monkey at each vertex and have each monkey to throw a dart. If it throws to the left, assign the vertex to V_1, and if it throws to the right, assign it to V_2. More formally, given a graph $G = (V, E)$, we assign each vertex independently with equal probability to either V_1 or V_2. This will give us a cut C of G, and we will show that the expected size of $C \geq |C^*|/2$.

Consider an edge $(i, j) \in E$. $\Pr[(i, j) \in C] = 1/2$. For $e \in E$, define χ_e to be a random

variable such that $\chi_e = 1$ if $e \in C$ and $\chi_e = 0$ if $e \notin C$. Then $|C| = \sum_{e \in E} \chi_e$. Thus,

$$\mathrm{E}[|C|] = \sum_{e \in E} \mathrm{E}[\chi_e] = \sum_{e \in E} \Pr[e \in C] = \frac{|E|}{2} \geq \frac{1}{2}|C^*|.$$

The first equality follows from linearity of expectation, $E[X + Y] = E[X] + E[Y]$, for any two random variables X and Y. This formula holds even if X and Y are not independent.

5.2.3 Derandomizing MAXCUT Approximation Algorithm Using Universal Hash Functions

Let $G = (V, E)$ be a graph with $V = \{0, \ldots, n-1\}$. Set k so that $2^k \geq n > 2^{k-1}$. Choose a and b at random from $\mathrm{GF}(2^k)$. For each $i \in V$, treat i as a member of $\mathrm{GF}[2^k]$ compute $ai + b$. Assign i to either V_1 or V_2 according to the first bit of $ai + b$. We claim that the expected size of cut obtained is $|E|/2$.

Let $\chi_{(a,b)}(i) =$ the first bit of $ai + b$. We know that $\{ai + b\}_{a,b \in \mathrm{GF}(2^k)}$ is a universal family of hash functions. Thus, $\{\chi_{(a,b)}\}_{a,b \in \mathrm{GF}(2^k)}$ is a universal family of hash functions. Then, a cut C obtained by the above randomized algorithm is given by $C = \{(i,j) | \chi_{(a,b)}(i) \neq \chi_{(a,b)}(j)\}$.

Because $\{\chi_{(a,b)}\}_{a,b \in \mathrm{GF}(2^k)}$ is a universal family of hash functions, $\Pr[\chi_{(a,b)}(i) \neq \chi_{(a,b)}(j)] = 1/2$. Thus, using the analysis from Section 3, we have $\mathrm{E}[|C|] = |E|/2$.

We can derandomize the above algorithm in polynomial time. There are less than $4n^2$ different choices for (a, b). To derandomize, we can examine the cuts created by $\{\chi_{(a,b)}\}$ for all $a, b \in \mathrm{GF}(2^k)$ in polynomial time. One of these cuts is guaranteed to be at least $|C^*|/2$ because $\mathrm{E}[|C|] \geq |C^*|/2$. This gives us a deterministic r-approximation algorithm for MAXCUT.

Although this derandomized algorithm does not give a better approximation ratio than the greedy algorithm, it is a parallel algorithm. For each pair (a, b), the determination of which side of the cut each vertex is on is independent of the other vertices. Thus, this can be executed in parallel. Additionally, the cut produced from each pair (a, b) is independent of the cuts from other pairs. Therefore, all cuts can be determined in parallel. After these cuts have been computed, the maximum can be found with an NC computation.

5.2.4 Goemans-Williamson Algorithm

Goemans and Williamson gave an approximation algorithm for MAXCUT with error ratio of about 12%. Randomization plays an important role here combined with semidefinite optimization.

Typically a subset of $[n]$ can be described by a binary sequence in $\{0, 1\}^n$. Although this is not essential, we will find it here more convenient to describe such a subset $V_1 \subseteq V$,

which corresponds to a cut $V_1 \cup (V \setminus V_1)$, by a vector $x \in \{-1, 1\}^n$, by letting $x_i = -1$ iff $i \in V_1$. Then the cut size is $\sum_{e=\{i,j\}} (x_i - x_j)^2/4$. Therefore the MAXCUT problem seeks to maximize

$$\frac{1}{4} \sum_{e=\{i,j\}} (x_i - x_j)^2$$

subject to the constraint $x \in \{-1, 1\}^n$. This constraint can be expressed as

$$x_i^2 = 1, \forall i \in V.$$

Such a problem is called a quadratic programming problem, which, in its generality, is also NP-hard (which is of course no surprise, since we have just reduced an NP-hard problem MAXCUT to it.)

The next trick is to linearize the problem by introducing a set of new variables $y_{ij}, 1 \leq i, j \leq n$, with the intention that $y_{ij} = x_i x_j$. Under this new set of variables, the objective function becomes

$$\frac{1}{4} \sum_{e=\{i,j\}} (y_{ii} + y_{jj} - 2y_{ij})$$

to be maximized subject to

$$y_{ii} = 1, \forall i \in V.$$

Observe that a "solution" y_{ij} need not correspond to any real solution x_i. For example, if $y_{ij} = x_i x_j$, then the matrix $Y = (y_{ij})$ is symmetric and positive semi-definite, being the product of X and its transpose X^{T},

$$Y = XX^{\mathrm{T}} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} (x_1 x_2 \ldots x_n)$$

Not only that, since X is $n \times 1$, Y is of rank 1 (assuming $X \neq 0$, which is implied by $y_{ii} = 1$). However, we choose to ignore this rank condition, yet preserve the constraint that Y is symmetric and positive semi-definite. The reason for this is that there is a polynomial time algorithm, based on the ellipsoid method, which solves this "semi-definite" programming problem optimally.

What we have done is called a "relaxation" of the original problem. The requirement of Y being positive semi-definite amounts to a seemingly infinitely many inequalities

$$v^{\mathrm{T}} Y v \geq 0, \forall v \in \mathbf{R}^n.$$

These are linear constraints on y_{ij}. It is a trick of the "semi-definite" programming algorithm which can handle these infinitely many inequalities implicitly. We will not dwell further on this, and simply assume such a polynomial time algorithm is available.

Coming back to MAXCUT, in polynomial time we find a symmetric and positive semi-definite Y^*, with $y^*_{ii} = 1$ which maximizes $\frac{1}{4}\sum_{e=\{i,j\}}(y_{ii} + y_{jj} - 2y_{ij})$ among all Y satisfying these constraints. Let the maximum value be M^*. Since this is a relaxation of the original problem, M^* is an upper bound of the maximum cut size.

It is known that a symmetric and positive semi-definite matrix Y^* can be expressed as a product of UU^{T}, where $U^{\mathrm{T}} = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix}$, and where $u_i \in \mathbf{R}^n$. Moreover this decomposition can be found in polynomial time.

Thus, $y_{ii} = u_i^{\mathrm{T}} u_i = ||u_i||^2 = 1$ says that each vector u_i is a unit vector. Moreover, "formally" retracing the expansion of $\sum_{e=\{i,j\}}(x_i - x_j)^2$ into $\sum_{e=\{i,j\}}(y_{ii} + y_{jj} - 2y_{ij})$, we get

$$M^* = \frac{1}{4}\sum_{e=\{i,j\}} ||u_i - u_j||^2.$$

The next idea of Goemans and Williamson is brilliant. *Randomly* choose a hyperplane Π in $\mathbf{R}^n$. This amounts to choosing a unit vector $v \in \mathbf{R}^n$, uniformly on the unit sphere, as the normal vector to Π. (This can be carried out approximately with exponentially small error; we are ignoring issues of discretizing the process here.) Now partition V according to which side of Π the vector u_i falls. More precisely, assign vertex i to V_1 iff the inner product $\langle u_i, v\rangle > 0$.

If we express the cut size thus formed as a sum of 0-1 random variables, which indicate whether edge $e = \{i, j\}$ belongs to the cut, then the expectation is

$$\sum_{e=\{i,j\}} \Pr[\ \Pi \text{ separates } u_i, u_j].$$

To investigate this probability $\Pr[\ \Pi$ separates $u_i, u_j]$, we only need to think of it in terms of 2-dimensional space spanned by u_i and u_j. Clearly this probability is θ_{ij}/π where θ_{ij} is the angle between u_i and u_j. Meanwhile the length $||u_i - u_j||$ is clearly $2\sin\frac{\theta_{ij}}{2}$.

Simple calculus shows that the function

$$f(\theta) = \frac{\theta}{4}\sin^2\frac{\theta}{2}$$

achieves minimum .69002507... at 2.331122370.... It follows that the expectation of the cut

$$\sum_{e=\{i,j\}} \frac{\theta_{ij}}{\pi} \geq \frac{.69002507}{\pi} \sum_{e=\{i,j\}} ||u_i - u_j||^2 \approx 0.878 \cdot M^*.$$

This is a randomized MAXCUT algorithm that achieves 87.8% approximation ratio. There are ways to derandomize this algorithm, but we will not discuss this problem any further.

5.3 Randomized Algorithm: Two Stage Hashing

One simple application of universal hashing is a Dctionary data structure by two stage hashing developed by Fredman, Komlos, and Szemeredi (FKS). We will define a quantity called the collision number, use no more complicated a tool as the Markov inequality to prove its properties. The collision number is generally useful elsewhere as well.

Given a finite set U and a subset $N \subseteq U$, we want to build an efficient dictionary. A dictionary is a data structure where we store information about the elements of N. Then, upon receiving a query $q \in U$, we want to look-up the dictionary and tell whether $q \in N$ or not efficiently. For example, let U be the set of all "words" of length five over the English alphabet and let N be the set of all proper English words of length five. This is where the name dictionary comes about. Various performance criteria of interest are time required to build the dictionary, space occupied by the dictionary and look-up time.

There are many deterministic schemes to implement a dictionary. These are usualy based on some balanced tree structure, and typically take $O(\log n)$ steps for each operation. Yao was the first to question this fundamental methodology ("Should tables be sorted") and proposed hashing as an efficient alternative. The FKS two-stage hashing scheme is extremely simple. It shows that given a universe U and a subset $N \subseteq U$, where $n = |N|$, one can build a dictionary for N that has the following properties:

- The dictionary can be built in $O(n)$ expected time.
- The dictionary occupies $O(n)$ space.
- Dictionary lookup takes $O(1)$ time.

Let us choose a set T to be our hash table. Size of the table $|T|$ will be fixed later. We choose a suitable hash function $h : U \mapsto T$ and for each element $u \in U$, hash it to the bucket numbered $h(u)$ in T. Then, given a query $q \in U$, we can check $q \in N$ or not by scanning the bucket given by $h(q)$. We say that elements $a, b \in N$ collide under h, if $h(a) = h(b)$. It is clear that more the number of collisions, it will take more time to look-up. Thus, the goal is to choose a good hash function h that minimizes the number of collisions, and yet does not use too much space.

Let H be a universal family of hash functions from U to T. Recall that such a family satisfies the following property. For any $u_1, u_2 \in U$ with $u_1 \neq u_2$,

$$\Pr_{h \in H}[h(u_1) = h(u_2)] = \frac{1}{|T|}.$$

Choose a hash function h at random from H. Let C be the random variable that gives the number of collisions under h. For $a, b \in N$, $a \neq b$, let $X_{a,b}$ be the indictor variable that denotes whether a and b collide under h (i.e. $X_{a,b} = 1$, if $h(a) = h(b)$ and 0, otherwise). Then,

$$C = \sum_{\substack{\{a,b\} \subset N \\ a \neq b}} X_{a,b}$$

Since the probability that a pair (a, b), where $a \neq b$, collides under a random hash function h is $\frac{1}{|T|}$, the expected number of collisions is

$$E(C) = \binom{n}{2} / |T|$$

Set $|T| = n^2$. Then, $E(C) < \frac{1}{2}$. By Markov's inequality, $\Pr[C \geq 1] \leq \frac{1}{2}$. So, $\Pr[C < 1] \geq \frac{1}{2}$. Since C is a integer variable, $C < 1$ implies that $C = 0$. Hence, if we choose h at random, with probability at least $1/2$, there will be no collisions under h. We can try various h at random until we find a h with $C = 0$. The expected number of trials needed is just 2. This scheme is wonderful, except that, we have to allocate $O(n^2)$ space to get this bound with a single random hash function. We can do better by using the following two-stage hashing scheme.

In the first stage of the scheme, we allocate only $|T| = n$. Then, for a random $h \in H$, $E(C) < \frac{n}{2}$. Using Markov's inequality again, $\Pr[C \geq n] \leq \frac{1}{2}$. By trying various hash functions at random, we can find a hash function with $C \leq n$. The expected number of trial will be only 2. Suppose we have found a hash function h_0 with the number of collisions $C_0 \leq n$. Next we proceed to the second stage of the scheme.

Let N_i be the set of elements of N that got mapped to i^{th} bucket of T under h_0. Let $n_i = |N_i|$. In the second stage, for each bucket $1 \leq i \leq |T|$, we build a hash table T_i for the set N_i, with $|T_i|$ to be n_i^2. Fix a universal family of hash functions $H_i : U \mapsto T_i$. Then, for a hash function h chosen at random from H_i, the expected number of collisions is

$$E(C) = \binom{n_i}{2} \frac{1}{|T_i|} = \binom{n_i}{2} \frac{1}{n_i^2} < 1/2.$$

Thus, $\Pr[C > 1] \leq 1/2$, which implies $\Pr[C = 0] \geq 1/2$. For each i, by trying various h at random, with an expected number of two trials, we can find a h_i with no collisions. Thus in total we expect to try no more than $2n$ times.

What will be the total size of the hash table (both stages combined)? It is given by

$$\text{Total size} = |T| + \sum_{i=1}^{|T|} |T_i| = n + \sum_{i=1}^{n} n_i^2.$$

We now need a bound for the summation. We obtain a bound by using the fact that the number of collisions in first stage $C_0 \leq n$. We express C_0 in a different way to get the bound.

$$\begin{aligned} C_0 &= \sum_{i=1}^{n} \binom{n_i}{2} \\ &= \frac{1}{2}\left(\sum_{i=1}^{n} n_i^2 - \sum_{i=1}^{n} n_i\right) \end{aligned}$$

Note that $\sum_{i=1}^{n} n_i = |N| = n$. Thus,

$$\sum_{i=1}^{|T|} n_i^2 = 2C_0 + \sum_{i=1}^{n} n_i \leq 2n + n = O(n).$$

So, the total space needed to store the main hash table T and the second level tables is only $O(n)$ (We will need space to store the hash functions h_0 and various chosen h_i. But this needs only $O(\log |U|)$ space, which we assume to be negligible). We next compute time required to build the tables. In the first stage, expected number of trials to find a "good" hash function h_0 is 2. Given a hash function h we can tell whether it is good or bad in time $O(n)$. Thus, expected time for first stage is $O(n)$. In the second stage, for each bucket i, expected number of trials to find a good h_i is 2 and the time needed to tell whether a h is good or bad is $O(n_i)$. Thus, total expected time in second stage is $O(n_1) + O(n_2) + \cdots + O(n_{|T|}) = O(n)$.

Now let us turn to look-up operation. Given a query q, apply h_0 to q to get the relevant bucket $i = h_0(q)$, then apply the second level hash function h_i to q and scan the bucket $h_i(q)$ in table T_i. As each h_i is chosen to be collision-free, there will only one entry in the bucket $h_i(q)$. If this entry is q, then report that $q \in N$. If it is not, report that $q \notin N$. This process takes $O(1)$ time since we simply apply hash functions (We can construct universal family of hash functions, where computing any $h(x)$ takes time $O(\log |U|)$, which is assumed to be negligible).

5.4 Randomized Complexity Classes

5.4.1 Definitions

Definition 5.10 (BPP) *BPP stands for bounded error probabilistic polynomial time. A language L is in* BPP*, if there is a boolean predicate $D(\cdot, \cdot)$, computable in deterministic polynomial time such that,*

$$\begin{aligned} x \in L &\implies \Pr_y[D(x, y) = 1] \geq 3/4 \\ x \notin L &\implies \Pr_y[D(x, y) = 1] \leq 1/4, \end{aligned}$$

where the number of random bits used, $|y|$ is polynomial in length of the input $|x|$.

DEFINITION 5.11 (RP) *RP was the earliest version of such classes, and simply stands for randomized polynomial time. A language L is in* RP*, if there is a boolean predicate $D(\cdot,\cdot)$, computable in deterministic polynomial time such that,*

$$\begin{aligned} x \in L &\implies \Pr_y[D(x,y)=1] \geq 1/2 \\ x \notin L &\implies \Pr_y[D(x,y)=1] = 0, \end{aligned}$$

where the number of random bits used, $|y|$ is polynomial in length of the input $|x|$.

DEFINITION 5.12 (ZPP) *A language L is said to be in* ZPP *if there is a polynomial time computable function $D : \{0,1\}^* \times \{0,1\}^* \mapsto \{0,1,?\}$ such that if $x \in L$, for any y, its output $D(x,y) \in \{1,?\}$, and if $x \notin L$, for any y, its output $D(x,y) \in \{0,?\}$. Moreover, D should have high success probability:*

$$\begin{aligned} x \in L &\implies \Pr_y[D(x,y)=1] \geq 1/2 \\ x \notin L &\implies \Pr_y[D(x,y)=0] \geq 1/2, \end{aligned}$$

where the number of random bits used, $|y|$ is polynomial in length of the input $|x|$.

5.4.2 Amplification of BPP

In this section, we show that the probability of success of a BPP algorithm can be "amplified" to be as high as exponentially close to 1, with only a polynomial amount of extra work. We start with the Chernoff bound in Theorem 5.4.

Let L be a language accepted by a probabilistic polynomial time TM M, in the following sense:

$$\begin{aligned} x \in L &\implies \Pr[M(x;r)=1] \geq \frac{1}{2} + \epsilon \\ x \notin L &\implies \Pr[M(x;r)=1] \leq \frac{1}{2} - \epsilon \end{aligned}$$

for random strings r of an appropriate polynomial length in $n = |x|$. Here, ϵ can be as low as $1/p(n)$ for some fixed polynomial $p(\cdot)$. We wish to amplify the success probability of the algorithm. In particular, we want to get exponentially close to 1, meaning, for any fixed polynomial $q(\cdot)$, we want a machine M' with

$$\begin{aligned} x \in L &\implies \Pr[M'(x;r')=1] \geq 1 - e^{-q(n)} \\ x \notin L &\implies \Pr[M(x;r)=1] \leq e^{-q(n)} \end{aligned}$$

We require that M' run in polynomial time, and hence the length of the random string $|r'|$ used by M' should also be polynomially bounded.

The idea is to run M a large polynomial number of times, and take the majority vote. Given an input x, M' will simply run the machine M on input x some $2m+1$ times, with $m \geq q(n)/(4\epsilon^2)$, say, and accept x iff at least $m+1$ runs of M on x accept. In this process we will need independent and uniformly chosen random $r_1, r_2, \cdots, r_{2m+1}$, with a total length of the random string $|r'| = O(|r|q(n)/\epsilon^2)$.

Let X_i be the 0-1 random variable indicating the i-th run $M(x; r_i)$, $p = \Pr[M(x; r) = 1]$, and $S = \sum_i (X_i - p)$. Suppose $x \notin L$, then $p \leq \frac{1}{2} - \epsilon$. Apply Theorem 5.4 with $\Delta = (2m+1)\epsilon$, we get

$$\Pr[M'(x) \text{ accepts }] = \Pr[\sum_i X_i \geq m+1] \leq \Pr[S \geq (2m+1)\epsilon] \leq e^{-q(n)}.$$

Similarly, if $x \in L$, then $p \geq \frac{1}{2} + \epsilon$, and

$$\Pr[M'(x) \text{ rejects }] \leq e^{-q(n)}.$$

When ϵ^{-1} is polynomially bounded, so is $m = O(q(n)/\epsilon^2)$. This achieves exponentially small error probability.

5.5 Sipser-Lautemann Theorem: BPP $\subseteq$ PH

Let L be a language in BPP. We will show that $L \in \Sigma_2^p$. Without loss of generality, there is a polynomial time computable predicate $D(\cdot, \cdot)$ such that,

$$\begin{aligned} x \in L &\implies \Pr_{y \in \{0,1\}^m}[D(x, y) = 1] \geq 1 - 1/m \\ x \notin L &\implies \Pr_{y \in \{0,1\}^m}[D(x, y) = 1] \leq 1/m \end{aligned}$$

where the number of random bits used m is polynomially bounded in input length $n = |x|$. For any input $x \in \{0,1\}^n$, define its witness set $W_x = \{y \in \{0,1\}^m | D(x, y) = 1\}$. So, if $x \in L$ then the witness set W_x is "fat", whereas, if $x \notin L$, the witness set is "thin". We will show how to test whether W_x is fat or thin in Σ_2^p and thereby prove that $L \in \Sigma_2^p$. Let us first formalize the notion of fat and thin sets and prove some properties of such sets.

We say that $S \subseteq \{0,1\}^m$ is *fat*, if $|S|/2^m \geq 1 - 1/m$. S is said to be thin, if $|S|/2^m \leq 1/m$. In general a subset S may be neither fat nor thin. But, the witness sets we are interested are always either fat or thin. For a string $u \in \{0,1\}^m$, define $S \oplus u = \{s \oplus u | s \in S\}$. Here, $s \oplus u$ denotes the m-bit string obtained by bit-wise XOR of s and u: if $s = s_1 s_2 \ldots s_m$ and $u = u_1 u_2 \ldots u_m$, then $s \oplus u = s_1 \oplus u_1 \cdots s_m \oplus u_m$. Think of $\{0,1\}^m$ as a vector space, S as a subset of this space and u to be a vector in it. Then, $S \oplus u$ is nothing but the subset obtained by "shifting" S by u.

We first discuss informally the effect of shifting fat and thin sets. Suppose S is fat. Then, if we choose a suitable number of shift vectors $u_1, u_2, \ldots, u_r$ at random, with high probability, the union of these shifts will "cover" the entire space: $\bigcup_{i=1}^{r}(S \oplus u_i) = \{0,1\}^m$. On the other hand, if S is a thin, then for any set of vectors $u_1, u_2, \ldots, u_r$, where $r < m$, the shifts will not cover the space: $\bigcup_{i=1}^{r}(S \oplus u_i) \neq \{0,1\}^m$. We will formally prove these claims. Observe that, with these claims, it is easy to put L in Σ_2^p: the condition "there is a set of vectors such that W_x shifted by these vectors covers the entire space" can be expressed as a Σ_2^p predicate!

LEMMA 5.13 *Let $S \subseteq \{0,1\}^m$.*

1. *If S is thin, then for any $r < m$, for any set of r vectors, the shifts cannot cover the entire space:*
$$\Pr_{u_1,u_2,\ldots,u_r \in \{0,1\}^m}\left[\bigcup_{i=1}^{r}(S \oplus u_i) = \{0,1\}^m\right] = 0.$$
2. *If S is fat, then with high probability, randomly chosen shifts will cover the entire space:*
$$\Pr_{u_1,u_2,\ldots,u_r \in \{0,1\}^m}\left[\bigcup_{i=1}^{r}(S \oplus u_i) = \{0,1\}^m\right] \geq 1 - \frac{2^m}{m^r}$$

Proof. The first part is obvious. For any vectors $u_1, u_2, \ldots, u_r$, the union of the shifts has cardinality,
$$\left|\bigcup_{i=1}^{r}(S \oplus u_i)\right| \leq r \cdot |S|$$
Since we assume that $r < m$ and S is thin, $|\bigcup_{i=1}^{r}(S \oplus u_i)| < 2^m$. Thus, the shifts cannot cover the entire space $\{0,1\}^m$ which has cardinality 2^m.

To prove the second part, we will bound probability of the negation of the event under consideration. A set of vectors $u_1, u_2, \ldots, u_r$ do not cover the entire space means that some vector $y \in \{0,1\}^m$ is not covered by these shifts. So,
$$\Pr_{u_1,u_2,\ldots,u_r}\left[\bigcup_{i=1}^{r}(S \oplus u_i) \neq \{0,1\}^m\right] \leq \sum_{y \in \{0,1\}^m} \Pr_{u_1,u_2,\ldots,u_r}\left[y \notin \bigcup_{i=1}^{r}(S \oplus u_i)\right]$$
Fix any $y \in \{0,1\}^m$. By the properties of $\oplus$ function, we see that $r \notin S \oplus u$ iff $r \oplus u \notin S$ (we use the fact that for any u, $u \oplus u = 0$). Thus, y will not be covered by the r shifts iff $\{y \oplus u_1, y \oplus u_2, \ldots, y \oplus u_r\} \cap S = \emptyset$. For any u chosen uniformly at random from $\{0,1\}^m$, $y \oplus u$ is distributed uniformly in $\{0,1\}^m$. (We use the fact that the function $f_y : u \mapsto y \oplus u$ is 1-1.) So, for a randomly chosen u, $\Pr[y \oplus u \notin S] \leq 1/m$, because S is fat. By independence, it follows that
$$\Pr_{u_1,u_2,\ldots,u_r}[\{y \oplus u_1, y \oplus u_2, \ldots, y \oplus u_r\} \cap S = \emptyset] = \prod_{i=1}^{r} \Pr_{u_i}[y \oplus u_i \notin S] \leq \left(\frac{1}{m}\right)^r.$$

We conclude that,

$$\begin{aligned}\Pr_{u_1,u_2,\ldots,u_r}\left[\bigcup_{i=1}^{r}(S\oplus u_i)\neq\{0,1\}^m\right] &\leq \sum_{y\in\{0,1\}^m}\Pr_{u_1,u_2,\ldots,u_r}\left[y\notin\bigcup_{i=1}^{r}(S\oplus u_i)\right]\\ &= 2^m\cdot\left(\frac{1}{m}\right)^r\\ &= \frac{2^m}{m^r}.\end{aligned}$$

Part 2 of the lemma follows from the above bound. ♣

Using Lemma 5.13, we can show that BPP $\subseteq \Sigma_2^p$. For a suitable value of r, the lemma will show that if S is fat, then for some set of r vectors, the shifts would cover the entire space, and if S is thin, then for any set of r vectors, the shifts would not cover the entire space. The above property can be tested in Σ_2^p. Formal proof is given below.

THEOREM 5.14 (SIPSER–LAUTEMANN) BPP $\subseteq \Sigma_2^p$.

Proof. Let $L \in$ BPP. Without loss of generality, there is a polynomial time computable predicate $D(\cdot,\cdot)$ such that,

$$\begin{aligned}x\in L &\implies \Pr_{y\in\{0,1\}^m}[D(x,y)=1]\geq 1-1/m\\ x\notin L &\implies \Pr_{y\in\{0,1\}^m}[D(x,y)=1]\leq 1/m\end{aligned}$$

where the number of random bits used m is polynomially bounded in input length $n = |x|$. For an input $x \in \{0,1\}^n$, define its witness set $W_x = \{y | D(x,y) = 1\}$. If $x \in L$ then the witness set W_x is fat, whereas, if $x \notin L$, the witness set is thin. Choose $r = m/2$. Then, from Lemma 5.13, we have

$$\begin{aligned}x\in L &\implies \Pr_{u_1,u_2,\ldots,u_r}\left[\bigcup_{i=1}^{r}(W_x\oplus u_i)=\{0,1\}^m\right]\geq 1-\frac{2^m}{m^r}\\ x\notin L &\implies \Pr_{u_1,u_2,\ldots,u_r}\left[\bigcup_{i=1}^{r}(W_x\oplus u_i)=\{0,1\}^m\right]=0.\end{aligned}$$

For sufficiently large m, the first probability > 0. So, if $x \in L$, there exists r vectors such that the shifts cover the entire space, and if $x \notin L$, for any r vectors, the shifts do not cover the entire space. Observe that, for any $u_1, u_2, \ldots, u_r$,

$$\bigcup_{i=1}^{r}(S\oplus u_i)=\{0,1\}^m \iff \forall y\in\{0,1\}^m\,[\vee_{i=1}^{r}(y\oplus u_i\in W_x)]$$

It follows that,

$$x \in L \iff \exists u_1, u_2, \ldots, u_r \forall y \in \{0,1\}^m \left[\vee_{i=1}^r (y \oplus u_i \in W_x)\right].$$

$y \oplus u_i \in W_x$ simply means that $D(x, y \oplus u_i) = 1$. So we conclude that,

$$x \in L \iff \exists u_1, u_2, \ldots, u_r \forall y \in \{0,1\}^m \left[\vee_{i=1}^r (D(x, y \oplus u_i) = 1)\right].$$

The predicate $\vee_{i=1}^r (D(x, y \oplus u_i) = 1)$ is testable in polynomial time, as r is polynomial in n and D is a polynomial time predicate. We conclude that $L \in \Sigma_2^p$. ♣

BPP is closed under complementation. So, we have also shown that BPP $\in \Pi_2^p$. We can also prove this claim directly by exhibiting a Π_2^p predicate. To do that, we first rephrase Lemma 5.13. Observe that, for any set $S \subseteq \{0,1\}^m$ and $u \in \{0,1\}^m$, $(S \oplus u)^c = S^c \oplus u$. Hence, for any $u_1, u_2, \ldots, u_r$,

$$\left[\bigcup_{i=1}^r (S \oplus u_i) = \{0,1\}^m\right] \iff \left[\bigcap_{i=1}^r (S^c \oplus u_i) = \emptyset\right].$$

Moreover, if S is thin then S^c is fat. So, Lemma 5.13 can be rephrased as follows. Suppose S is fat. Then S^c is thin. Applying Part 1 of the lemma to S^c, we see that for any set of $r < m$ shift vectors, $u_1, u_2, \ldots, u_m$, the intersection of the shifts is non-empty. Now suppose S is thin. Then S^c is fat. Applying Part 2 of the lemma to S^c, we see that for randomly chosen r shift vectors $u_1, u_2, \ldots, u_m$, with high probability, the intersection of the shifts is empty. Formally,

LEMMA 5.15 *Let $S \subseteq \{0,1\}^m$.*

1. *If S is fat, then for any $r < m$, for any set of r vectors, the intersections of the shifts is non-empty:*
$$\Pr_{u_1, u_2, \ldots, u_r}\left[\bigcap_{i=1}^r (S \oplus u_i) \neq \emptyset\right] = 1.$$
2. *If S is thin, then with high probability, randomly chosen shifts will have an empty intersection:*
$$\Pr_{u_1, u_2, \ldots, u_r}\left[\bigcap_{i=1}^r (S \oplus u_i) = \emptyset\right] \geq 1 - \frac{2^m}{m^r}$$

Applying Lemma 5.15 to witness sets, with $r = m/2$, we have

$$x \in L \implies \forall u_1, u_2, \ldots, u_m \left[\bigcap_{i=1}^r (W_x \oplus u_i) \neq \emptyset\right]$$

$$x \notin L \implies \exists u_1, u_2, \ldots, u_m \left[\bigcap_{i=1}^r (W_x \oplus u_i) = \emptyset\right]$$

The above property can be expressed as a Π_2^p predicate:

$$x \in L \iff \forall u_1, u_2, \ldots, u_m \exists y \left[\wedge_{i=1}^r (y \oplus u_i) \in W_x)\right]$$

5.6 Isolation Lemma

The isolation lemma provides a mechanism to approximately compute the size of some set $S \subseteq \{0,1\}^m$, such as a BPP witness set. In this section, we state and prove the lemma. In the subsequent sections, we use the lemma to give an alternative proof that BPP $\subseteq \Sigma_2^p$, and also discuss approximate counting.

Let S and T be finite sets and $\mathcal{H}$ be a universal family of hash functions from S to T. Two distinct elements $x, y \in S$ are said to *collide* under a hash function $h \in \mathcal{H}$, if $h(x) = h(y)$. We say that a hash function h *isolates* an element $x \in S$, if no element in S collides with x. One can imagine that h "likes" x and gives it a separate seat in T to sit alone comfortably! A set of hash functions $\{h_1, h_2, \ldots, h_r\}$ is said to isolate an element $x \in S$, if one of these function h_i isolates x. The set of functions is said to *isolate all of* S, if for every element $x \in S$, there is some function h_i in the set that isolates x.

Suppose we choose r hash functions $h_1, h_2, \ldots, h_r$, uniformly and independently at random from $\mathcal{H}$. What is the probability that the set of these functions isolates all of S? The answer depends on the size of S compared to the size of T and the value of r. Intuitively, if S is sufficiently smaller than T and h is large enough, with high probability, the randomly chosen set of functions will isolate all of S. On other hand, if S is large compared to T and r is small enough, then there is not much space in T to give "separate seats" for all the elements in S. For suitable choices of $|T|$ and r, we can make the probability of isolation zero.

Lemma 5.16 (Isolation Lemma) *Let $\mathcal{H}$ be a universal family of hash functions from S to T. Let $h_1, h_2, \ldots, h_r$ be chosen uniformly and independently at random from $\mathcal{H}$, where $r > 1$.*

1. *If $|S| \geq r|T|$, then*

$$\Pr_{h_1,h_2,\ldots,h_r}[\{h_1, h_2, \ldots, h_r\} \text{ isolates all of } S] = 0.$$

2. *Suppose $|S|^{r+1} \leq |T|^r$. Then,*

$$\Pr_{h_1,h_2,\ldots,h_r}[\{h_1, h_2, \ldots, h_r\} \text{ isolates all of } S] > 1 - \frac{|S|^{r+1}}{|T|^r}.$$

Proof. The first part of lemma is quite obvious. Any hash function $h \in \mathcal{H}$ can isolate at most $|T| - 1$ elements in S (it can assign "separate seats" for some $|T| - 1$ elements in S and then map all the other elements to the last "seat" in T). So, any set of r hash functions can together isolate at most $r(|T| - 1)$ elements. As $|S| \geq r|T|$, no set of r hash functions can isolate all of S.

We next prove the second part of the lemma. Fix any distinct elements $x, y \in S$. From the properties of universal family of hash functions, the probability that x and y collide under h is $1/|T|$. So, for any element $x \in S$,

$$\begin{aligned} \Pr_{h\in\mathcal{H}}[h \text{ does not isolate } x] &\leq \sum_{y\in S-\{x\}} \Pr_{h\in\mathcal{H}}[x \text{ and } y \text{ collide under } h] \\ &< \frac{|S|}{|T|} \end{aligned}$$

It follows that, for any x in S, if we choose $h_1, h_2, \ldots, h_r$ uniformly and independently at random,

$$\Pr_{h_1,h_2,\ldots,h_r}[\text{none of the } h_i \text{ isolate } x] < \left(\frac{|S|}{|T|}\right)^r.$$

Finally, the probability that randomly chosen r hash functions do not isolate all of S can be bounded by summing up over all possible elements in S:

$$\begin{aligned} &\Pr_{h_1,h_2,\ldots,h_r\in\mathcal{H}}[\{h_1, h_2, \ldots, h_r\} \text{ does not isolate all of } S] \\ \leq\ &\sum_{x\in S} \Pr_{h_1,h_2,\ldots,h_r\in\mathcal{H}}[\text{none of } h_1, h_2, \ldots, h_r \text{ isolate } x] \\ <\ &|S| \times \left(\frac{|S|}{|T|}\right)^r \end{aligned}$$

We conclude that

$$\Pr_{h_1,h_2,\ldots,h_r\in\mathcal{H}}[\{h_1, h_2, \ldots, h_r\} \text{ isolates all of } S\ ] > 1 - \frac{|S|^{r+1}}{|T|^r}$$

♣

5.7 BPP $\subseteq \Sigma_2^p$ - Another Proof Using the Isolation Lemma

This section shows how to use the isolation lemma to put BPP in Σ_2^p. We first present the proof idea informally. Let L be a language in BPP, via a randomized algorithm that uses m random bits, where m is polynomial in the input length. We assume that the algorithm has been suitably amplified. We will fix parameters r and size of the target set T suitably in Lemma 5.16. For any input x, let $W_x \subseteq \{0,1\}^m$ be the set of witness strings on which the algorithm accepts the input x. The parameters will be chosen appropriately, so that if $x \in L$, W_x will be large enough that no set of r hash functions $h_1, h_2, \ldots, h_r$ will isolate all of W_x; on the other hand, if $x \notin L$, W_x will be small enough so that, with high probability, randomly chosen r hash functions $h_1, h_2, \ldots, h_r$ will isolate all of W_x, and in particular there will exist some set of r hash functions that isolates all of W_x. Hence, $x \notin L$ iff there exist a set of r hash functions $\{h_1, h_2, \ldots, h_r\}$ that isolates all of W_x. The later condition can be expressed as a Σ_2^p predicate, thereby placing L in Π_2^p. A formal proof follows.

THEOREM 5.17 $\text{BPP} \subseteq \Sigma_2^p \cap \Pi_2^p$.

Proof. Let $L \in \text{BPP}$. We prove that $L \in \Pi_2^p$. As BPP is closed under complementation, the lemma follows. Without loss of generality, there is a deterministic polynomial time boolean predicate $D(\cdot, \cdot)$ such that for any input $x \in \{0,1\}^n$,

$$\begin{aligned} x \in L &\implies \Pr_{y \in \{0,1\}^m}[D(x,y) = 1] \geq \frac{1}{2} \\ x \notin L &\implies \Pr_{y \in \{0,1\}^m}[D(x,y) = 1] \leq \frac{1}{4m} \end{aligned}$$

where the number of random bits used m is polynomial in the input length n. Wolog we can assume m is a power of 2. Fix $r = m$, and let T be a set of size $2^m/(2m)$. (As it is usually the case in such proofs, the above choices of parameters such as the extent of amplification, value of r and size of $|T|$ are not crucial. We can fix them in many ways to make the proof work! One such setting is given above). Let $\mathcal{H}$ be a universal family hash functions from $\{0,1\}^m$ to T. For any input x, let $W_x = \{y | D(x,y) = 1\}$. Suppose $x \in L$. Then,

$$|W_x| \geq 2^{m-1} = r \cdot |T|.$$

So, Lemma 5.16 shows that no set of r hash functions $\{h_1, h_2, \ldots, h_r\}$ from $\mathcal{H}$ isolates all of W_x. On the other hand, suppose $x \notin L$. Then, from our choice of parameters,

$$\frac{|W_x|^{r+1}}{|T|^r} \leq \frac{1}{4m}.$$

Again, Lemma 5.16 implies that if we choose $h_1, h_2, \ldots, h_r$ uniformly and independently at random from $\mathcal{H}$,

$$\Pr_{h_1, h_2, \ldots, h_r}[\{h_1, h_2, \ldots, h_r\} \text{ isolates } W_x] \geq 1 - \frac{1}{4m}.$$

In particular, it follows that,

$$\begin{aligned} x \in L &\implies \forall h_1, h_2, \ldots, h_r [\{h_1, h_2, \ldots, h_r\} \text{ does not isolate } W_x] \\ x \notin L &\implies \exists h_1, h_2, \ldots, h_r [\{h_1, h_2, \ldots, h_r\} \text{ isolates } W_x] \end{aligned}$$

So, we have,

$$x \notin L \iff \exists h_1, h_2, \ldots, h_r [\{h_1, h_2, \ldots, h_r\} \text{ isolates } W_x].$$

Given particular $h_1, h_2, \ldots, h_r$, the predicate "$\{h_1, h_2, \ldots, h_r\}$ isolates W_x" can be expressed as

$$(\forall y \in W_x)(\exists 1 \leq i \leq r)(\forall y' \in W_x - \{y\})[h_i(y) \neq h_i(y')].$$

At first glance, it seems we need to write a Π_3^p predicate to express the above condition. But, we can do better and express it as a coNP predicate, because $\exists 1 \leq i \leq r$ is a bounded

quantifier and can be eliminated. This is a general principle in logic, but to be totally concrete, we have $\{h_1, h_2, \ldots, h_r\}$ isolates W_x if and only if

$$(\forall y \in W_x)\{(\forall y_1 \in W_x - \{y\})[h_1(y) \neq h_1(y_1)] \vee \ldots \vee (\forall y_r \in W_x - \{y\})[h_r(y) \neq h_r(y_r)]\}$$

which is equivalent to

$$(\forall y \in W_x)(\forall y_1, \ldots, y_r \in W_x - \{y\})[(h_1(y) \neq h_1(y_1)) \vee \cdots \vee (h_r(y) \neq h_r(y_r)].$$

We conclude that, $x \notin L$ if and only if

$$\begin{aligned}(\exists h_1, \ldots, h_r)(\forall y \in W_x)(\forall y_1, \ldots, y_r \in W_x - \{y\}) \\ [(h_1(y) \neq h_1(y_1)) \vee (h_2(y) \neq h_2(y_2)) \vee \cdots \vee (h_r(y) \neq h_r(y_r)].\end{aligned}$$

We have shown that $L \in \Pi_2^p$. ♣

5.8 Approximate Counting Using Isolation Lemma

Using Sipser's Isolation Lemma, Stockmeyer showed how to do approximate counting in $\mathrm{P}^{\Sigma_2^p}$. In fact, his techniques does approximate counting at the level of $\mathrm{RP}^{\mathrm{NP}}$ already. Let us be more precise.

Definition 5.18 (#P) *#P is a function class. A function $f : \{0,1\}^* \to \mathbf{N}$ is in #P, iff there is a p-time NTM M, such that $f(x) = \#$ of accepting paths of $M(x)$.*

So typically any NP langugae has a "counting" version; e.g., #SAT(ϕ) is the number of satisfying assignments to the formula ϕ; #HAM(G) is the number of Hamiltonian circuits in G. One can easily develop a notion of polynomial time reduction for these functions. It is no surprise that both functions #SAT and #HAM are #P-complete under this reduction. This follows from the fact that Cook's reduction is parsimonious, i.e., they preserve the number of solutions. (For instance, in Cook's reduction from a generic NP langugae to SAT, every accepting computation is in a unique way associated with a satisfying assignment.)

Less obvious, yet also #P-complete, is the permanent function: For any n by n matrix $A = (a_{ij})$,

$$\mathrm{per}(A) = \sum_{\sigma \in S_n} a_{1,\sigma 1} a_{2,\sigma 2} \ldots a_{n,\sigma n},$$

where the sum is over all permutations $\sigma : i \mapsto \sigma i$. In other words, the permanent function is defined much as the determinant function, except there are no more minus signs. For a 0-1 matrix, per(A) counts the number of perfect matchings of the bipartite graph with matrix A.

Valiant showed that the permanent function is also #P-complete (even though the decision problem of graph matching is in P.)

DEFINITION 5.19 (PP) *PP stands for probabilistic polynomial time. A language L is in* PP*, if there is a boolean predicate $D(\cdot,\cdot)$, computable in deterministic polynomial time such that,*

$$\begin{aligned} x \in L &\implies \Pr_{y\in\{0,1\}^m}[D(x,y)=1] \geq 1/2 \\ x \notin L &\implies \Pr_{y\in\{0,1\}^m}[D(x,y)=1] < 1/2, \end{aligned}$$

where the number of random bits used, m, is polynomial in length of the input $|x|$.

Note that, unlike BPP and RP, every polynomial time NTM defines a PP language, whereas not every polynomial time NTM defines a language in BPP or RP. BPP, RP and ZPP are "promise classes" in the sense that a polynomial time predicate (or equivalently a polynomial time NTM) defines a language in BPP, RP or ZPP only if it satisfies some global conditions. These conditions are, moreover, not decidable in general. Therefore we do not have an enumeration of these classes simply by an enumeration of their acceptors. This implies that we do not have a universal language, nor a complete language by this process. It is an open problem whether such complete languages exist. (Of course if BPP = P, then they indeed exist.) For PP, one can easily enumerate the class, and complete languages exist. For example, the set of Boolean formulae on n variables having at least 2^{n-1} satisfying assignments is such a language.

It is also easy to see that

THEOREM 5.20

$$\mathrm{P}^{\mathrm{PP}} = \mathrm{P}^{\#P}.$$

Next we discuss approximate counting. Given a formula φ over n variables, the goal is to approximately count the number of satisfying truth assignments of φ. Let $S = \{\sigma|\varphi(\sigma) = 1\}$ be the set of all such assignments. We want to compute the first $c \cdot \log n$ bits of $|S|$, where c is any constant. This can be done by a polynomial time algorithm using a Σ_2^p oracle, as shown by Stockmeyer. In fact, his technique can accomplish the same task with just an NP oracle, if we are ready to settle for a randomized polynomial time algorithm.

Our main tool is the isolation lemma, Lemma 5.16. We first rephrase the lemma in a more suitable format:

LEMMA 5.21 *Let $S \subseteq \{0,1\}^n$, and let $\mathcal{H}$ be a family of 2-universal hash functions from $\{0,1\}^n$ to $\{0,1\}^k$. For all $m \geq k$, choose $h_1, h_2, \ldots, h_m$ independently at random from $\mathcal{H}$.*

1. *if $|S| \leq 2^{k-1}$ then*

$$\Pr_{h_1,\ldots,h_m}[\forall x \in S \text{ some } h_i \text{ isolates } x\,] \geq 1 - \frac{1}{2^{m-k+1}}$$

2. if $|S| > m2^k$ *then*

$$\Pr_{h_1,\ldots,h_m}[\forall x \in S \text{ some } h_i \text{ isolates } x\,] = 0.$$

The idea is to try all values of k from 1 to n, and attempt to find a k such that $|S| \approx 2^k$. (We may assume our $S \neq \emptyset$; at any rate with one query to SAT we can verify this.) For any $\emptyset \neq S \subseteq \{0,1\}^n$, there is some k_S, where $1 \leq k_S \leq n$, such that $2^{k_S-1} \leq |S| \leq 2^{k_S}$. If we take every k in the range $1 \leq k \leq n+1$, and randomly pick $m = 2n$ hash functions $h_1, \ldots, h_m : \{0,1\}^n \to \{0,1\}^k$, then for each $k \geq k_S + 1$, we would get *isolation* with probability $\geq 1 - \frac{1}{2^n}$. For each k we ask the SAT oracle, whether the chosen set of $h_1, \ldots, h_m$ has the property that "$\forall x \in S$, one of h_i isolates x". Since there are only $m = 2n$ hash functions this is a SAT query. We pick the least k_0 such that the oracle confirms *isolation*. Then $k_0 \leq k_S + 1$, with probability $\geq 1 - \frac{1}{2^n}$. (We abort if for no k the chosen hash functions achieve *isolation*; this happens with exponentially small probability.) Also by the second part of Lemma 5.21, we know definitely $|S| \leq 2n2^{k_0}$. Denote by $U = 2n2^{k_0}$, then with high probability,

$$\frac{U}{8n} \leq |S| \leq U.$$

This give us a randomized polynomial time algorithm using a SAT oracle to approximate $|S|$ within $O(n)$. We'd like to do better. To do this, we use a little trick to amplify the accuracy.

First we build a set S' such that

$$S' = \underbrace{S \times S \times \cdots \times S}_{m \text{ times}} \quad \subseteq \{0,1\}^{nm}$$

where m is polynomial in n. Then we run the previous algorithm on the set S' to get an estimate U' for S' such that

$$\frac{U'}{8nm} \leq |S'| \leq U'.$$

Now we set $e(S) = (U')^{1/m}$. It follows that

$$\frac{e(S)}{(8nm)^{1/m}} \leq |S| \leq e(S).$$

By choosing m to be a sufficiently large polynomial in n, we can get

$$e(S) \cdot \left(1 - \frac{1}{n^c}\right) \leq |S| \leq e(S),$$

for any constant c.

5.9 Unique Satisfiability: Valiant–Vazirani Theorem

Assuming NP $\neq$ P, SAT cannot be solved in polynomial time. One may suspect that it is difficult to design a polynomial time algorithm for SAT because the input formula may have myriad truth assignments and it is hard to get hands on one of them. This suspicion leads to the following interesting problem, called USAT (Unique SAT). We are given a formula φ which is guaranteed to be either unsatisfiable or has exactly one satisfying truth assignment. Is USAT solvable in polynomial time? Here we show some evidence that it is unlikely. We prove that if USAT is solvable in polynomial time, then NP = RP.

THEOREM 5.22 *Suppose there is a polynomial time algorithm A, which for a given boolean formula φ, answers*

$$A(\varphi) = \begin{cases} \text{No} & \text{if } \varphi \text{ has no satisfying assignments;} \\ \text{Yes} & \text{if } \varphi \text{ has exactly one satisfying assignment;} \\ \text{Yes/No} & \text{if } \varphi \text{ has more than one satisfying assignments.} \end{cases}$$

Then NP = RP.

Proof. Note that the algorithm A may output anything (YES or NO) if it is given a formula with more than one satisfying truth assignment. Assuming A runs in polynomial time, we design a RP algorithm for SAT. The idea is to use some coin tosses and convert φ into more constrained formula φ', so that if φ is unsatisfiable, φ' will also be unsatisfiable. And if φ is satisfiable, with non-trivial probability, exactly one of the satisfying assignments of φ will satisfy φ'. Once we are successful in obtaining such a φ', we can apply the algorithm A to it.

To start with assume that φ is satisfiable and let $\#\varphi$ be the number of satisfying truth assignments of φ. In order to convert φ to φ', the algorithm needs an estimate for $\#\varphi$. It is computationally hard to estimate $\#\varphi$. So, we simply pick a k with $1 \leq k \leq n$ at random! The hope is that $2^{k-1} \leq \#\varphi \leq 2^k$. The probability that k satisfies the above condition is $1/n$, as long as $\#\varphi \neq 0$. Assume that we are lucky and k indeed satisfies the above condition. Next we choose a set T of size 2^{k+1}. If $2^{k-1} \leq \#\varphi \leq 2^k$, then $2(\#\varphi) \leq |T| \leq 4(\#\varphi)$, meaning $|T|$ is neither too big nor too small compared to $\#\varphi$. We then setup a universal family of hash functions H from the set of all assignments $\{0,1\}^n$ to T, and we randomly pick a hash function h from H and an element α from T. We will show that, with non-trivial probability, there will be a *unique* satisfying truth assignment x such that $h(x) = \alpha$. So, we consider the question: "Is there a truth assignment t such that $\varphi(t) = 1$ and $h(t) = \alpha$". By Cook's theorem this can be converted to a SAT question φ', and it will have a unique satisfying assignment iff there is a unique t satisfying φ, and is mapped to α by h. If we are lucky in choosing the "correct" k, h and α, φ' will have exactly one satisfying truth assignment.

Algorithm for SAT:
Input: A formula φ over n variables.

1. Choose a number $1 \leq k \leq n$ uniformly at random.
2. Let $T = \{0,1\}^{k+1}$, let H be a universal family of hash functions from $\{0,1\}^n$ to T.
3. Choose $h \in H$ and $\alpha \in T$ uniformly at random.
4. Let φ' be the Boolean formula from Cook's theorem, encoding the NP predicate "$(\exists t \in \{0,1\}^n)[(\varphi(t) = 1) \wedge (h(t) = \alpha)]$".
5. Run the procedure A for USAT on φ'. *Accept* φ iff $A(\varphi')$ = accept and the truth assignment extracted using A via self-reducibility indeed satisfies φ.

It is clear that, if φ is unsatisfiable then our algorithm will reject it with probability one. So, assume that φ is satisfiable. Then over the random choices of k, h and α, we show that our algorithm accepts φ with probability $\Omega(1/n)$. This probability can then be easily amplified, making it an RP algorithm.

First of all suppose $2^{k-1} \leq \#\varphi \leq 2^k$. The probability that k satisfies the above condition is $1/n$. Under this assumption, we have $2(\#\varphi) \leq |T| \leq 4(\#\varphi)$. We say that two *satisfying* distinct truth assignments a and b (with $a \neq b$) collide under h, if $h(a) = h(b)$. Over the random choice h, let C be the random variable that counts the number of collisions. We first show that the expected number of collisions $\mathrm{E}[C]$ is small. For any fixed $a \neq b$, the probability that a and b collide is $1/|T|$ (this follows from the definition of universal family of hash functions). For any pair of satisfying truth assignments $a \neq b$, let $X_{a,b}$ be a 0-1 random variable such that $X_{a,b} = 1$, if a and b collide under h and $X_{a,b} = 0$, if they don't collide. Its expectation is $\mathrm{E}[X_{a,b}] = 1/|T|$. The number of collisions C is the sum of over all $X_{a,b}$. So, expectation of C is

$$\mathrm{E}[C] = \sum_{(a,b)} \mathrm{E}[X_{a,b}].$$

The summation above ranges over all distinct pairs of satisfying truth assignments. The number of such pairs is $\binom{\#\varphi}{2}$. Hence,

$$\mathrm{E}[C] = \binom{\#\varphi}{2} \frac{1}{|T|} \leq \frac{\#\varphi}{4}.$$

Using Markov's inequality, we have

$$\Pr\left[C \geq \frac{\#\varphi}{3}\right] \leq \frac{E[C]}{\#\varphi/3} \leq \frac{3}{4}.$$

So, with probability at least $1/4$, the number of collisions C is at most $\#\varphi/3$. In general, if there are c collisions, at most $2c$ satisfying truth assignments can participate in collisions. (This can be shown easily by the inductive argument: Take any such x and pick any $y \neq x$ such that $h(x) = h(y)$. Now remove x and y. There can be at most $c-1$ collisions left among the remaining points, and thus at most $2c-2$ points.) Thus, in our case, at most

$2/3 \cdot \#\varphi$ satisfying truth assignments can participate in some collision. It follows that at least $\#\varphi/3$ satisfying truth assignments are mapped to a unique image. Call these $\geq \#\varphi/3$ images in T *good*: i.e., there is a unique satisfying assignment a such that $h(a) = t$. Recall that $|T| \leq 4(\#\varphi)$. So, at least $1/12$ fraction of elements in T are good. Therefore, with probability at least $1/12$, the randomly picked element α is good.

We can now lower bound the probability that φ' has a unique satisfying truth assignment. Assume that φ is satisfiable. Then, with probability $1/n$, the number k chosen by the algorithm satisfies $2^{k-1} \leq \#\varphi \leq 2^k$. Assuming k satisfies the above condition, with probability at least $1/4$, the number of collision C is at most $\#\varphi/3$. Assuming $C \leq \#\varphi/3$, with probability at least $1/12$, the randomly chosen α is good. If α is good, then φ' has exactly one satisfying truth assignment. Putting together,

$$\Pr_{k,h,\alpha}[\varphi' \text{ has a unique satisfying truth assignment}] \geq \frac{1}{n} \cdot \frac{1}{4} \cdot \frac{1}{12} = \frac{1}{48n}$$

As we noted already, if φ is unsatisfiable, our algorithm has zero probability of accepting it. If φ is satisfiable, our algorithm accepts it with probability at least $1/(48n)$. So our algorithm is an RP algorithm with success probability $1/(48n)$. We can amplify this success probability, as usual, by running the algorithm multiple times. For example, by running the algorithm $96n$ times, the success probability can be amplified to $1/2$. ♣

5.10 Efficient Amplification

Suppose we have some language L in RP. Thus membership of x in L can be determined by running over witness strings $y \in \{0,1\}^n$, for some $n = |x|^{O(1)}$, and achieving

$$\begin{aligned} x \in L &\implies \Pr_y[D(x,y) = 1] \geq 1/2 \\ x \notin L &\implies \Pr_y[D(x,y) = 1] = 0. \end{aligned}$$

A naive method of amplification of this probability can be obtained by running independent $y_1, \ldots, y_k \in \{0,1\}^n$. Thus, with kn random bits, we can achieve error probability $1/2^k$. A similar process with majority vote can be carried out for BPP langugaes, justified by the Chernoff bound. This probability is the ratio of the number of "bad" coin flips ($2^{k(n-1)}$) over the total number of coin flips (2^{kn}). Is there any way to achieve this reduction of error probability more efficiently?

5.10.1 Chor-Goldreich Generator

It turns out that there are several methods. Here is a first try. Let $\{h_s\}$ be a family of universal hash functions from $\{0,1\}^n \to \{0,1\}^n$. Now by randomly choosing s, we simply take the pairwise independent samples $y_i = h_s(i)$ as "witness" strings, for $i = 1, \ldots, k$. (We

consider $i = 1, \ldots, k$ as embedded in $\{0,1\}^n$, as long as $k \leq 2^n$.) For our RP langugae L, clearly if $x \notin L$ then all $D(x, y_i) = 0$. Suppose $x \in L$, we estimate the error probability that all $D(x, y_i) = 0$. We do this by using the Chebechev bound. Let Z_i be the 0-1 r.v. such that $Z_i = 1$ iff $D(x, y_i) = 1$.

THEOREM 5.23 *If $x \in L$, then*

$$\Pr_s[(\forall 1 \leq i \leq k) D(x, y_i) = 0] \leq 1/k.$$

The proof is a simple application of the Chebechev bound. Let $x \in L$. Then Z_i are pair-wise independent with expectation $\mu = \mathrm{E}[Z_i] = \Pr_y[D(x, y) = 1] \geq 1/2$, and as a 0-1 r.v. Z_i has variance $\leq 1/4$. Then

$$\Pr_s[(\forall 1 \leq i \leq k) D(x, y_i) = 0] = \Pr_s\left[\sum_{i=1}^{k} Z_i = 0\right] \leq \Pr_s\left[\left|\sum_{i=1}^{k} Z_i - k\mu\right| \geq k/2\right] \leq 1/k.$$

This is called the Chor-Goldreich generator, which achieves error $1/k$ with $2n$ bits. Of course one can not use this generator to achieve exponentially small error probability since this would require exponentially many evaluations of the hash function.

5.10.2 Hash Mixing Lemma and Nisan's Generator

The following generator of Nisan's was primarily devised for space bounded computations. But it can also be used for deterministic amplification.

Again let $\{h_s\}$ be a family of universal hash functions from $\{0,1\}^n \to \{0,1\}^n$. Pick uniformly and independently $s_1, \ldots, s_k$, and consider the following recursive definition of $G_i(y; s_1, \ldots, s_i)$, for $0 \leq i \leq k$. First $G_0(y) = y$, and for $i \geq 0$,

$$G_{i+1}(y; s_1, \ldots, s_{i+1}) = G_i(y; s_1, \ldots, s_i) \circ G_i(h_{s_{i+1}}(y); s_1, \ldots, s_i),$$

where $\circ$ denotes concatenation. Thus, $G_1(y; s_1) = y \circ h_{s_1}(y)$, $G_2(y; s_1, s_2) = y \circ h_{s_1}(y) \circ h_{s_2}(y) \circ h_{s_1}(h_{s_2}(y))$, etc. It may appear that in $G_1(y; s_1)$, it is not very clever since it uses $3n$ bits to produce only $2n$ (not even quite uniform) bits. The genius of this generator comes when you realize that this generator exponentially stretches its input seed. For G_k, it takes $(2k+1)n$ bits, and produces $2^k n$ bits. Also note that for any particular bit position, it is very simple to compute this bit in a random access manner, without recursion and without even having to compute the earlier bits sequentially. For G_k, the i th block of n bits can be directly obtained as follows. Let the k-bit binary representation of i be $i_k i_{k-1} \ldots i_1$, then the i th block of n bits is $h_{s_1}^{i_1} h_{s_2}^{i_2} \ldots h_{s_k}^{i_k}(y)$, where $h_{s_j}^1$ denotes h_{s_j} and $h_{s_j}^0$ denotes the identity function. With read-only access to the hash function seeds $s_1, s_2, \ldots, s_k$, this can be computed in space $O(k+n)$. These considerations will become important for its primary

intended purpose for space bounded computations; but they are not crucial for our purpose of amplification here.

How good are these bits? The following lemma is the technical tool to address this question.

LEMMA 5.24 (HASH MIXING LEMMA) *Let $\epsilon = 2^{-n/3}$. Then for all subset $E \subseteq \{0,1\}^n \times \{0,1\}^n$, for all but an $\epsilon/4$ fraction of s,*

$$| \Pr_{y \in \{0,1\}^n}[y \circ h_s(y) \in E] - \mu[E] | < \epsilon,$$

where, $\mu[E]$ is the probability measure of the set E, i.e., $\mu[E] = \Pr_{y,z \in \{0,1\}^n}[y \circ z \in E]$.

Proof. We want to estimate the fraction s such that it is "bad":

$$| \Pr_{y \in \{0,1\}^n}[y \circ h_s(y) \in E] - \mu[E] | \geq \epsilon. \tag{5.1}$$

For any fixed y we define the indicator random variable (random over s)

$$Z_y^{h_s} = \begin{cases} 1 & \text{if } y \circ h_s(y) \in E \\ 0 & \text{otherwise.} \end{cases}$$

We can write $\Pr_{y \in \{0,1\}^n}[y \circ h_s(y) \in E]$ as a sum of pair-wise independent random variables (random over s)

$$\frac{1}{2^n} \sum_{y \in \{0,1\}^n} \mathbf{1}_{[y \circ h_s(y) \in E]} = \frac{1}{2^n} \sum_y Z_y^{h_s},$$

then it has expectation

$$\frac{1}{2^n} \sum_{y \in \{0,1\}^n} \mu'[E_y] = \mu[E],$$

where E_y is the fibre-set over y, $E_y = \{z \mid y \circ z \in E\}$, and $\mu'[E_y]$ is the 1-dimensional probabiliy measure $\mu'[E_y] = |E_y|/2^n$.

Therefore we can use Chebechev inequality to estimate this deviation. Thus the "bad" event (5.1) over s has probability at most

$$\begin{aligned} \frac{1}{\epsilon^2} \mathbf{Var}\left[\frac{1}{2^n} \sum_y Z_y^{h_s}\right] &= \frac{1}{\epsilon^2} \frac{1}{2^{2n}} \sum_y \mathbf{Var}[Z_y^{h_s}] & (5.2) \\ &\leq \frac{1}{\epsilon^2 2^n 4}, & (5.3) \end{aligned}$$

where the first equality uses the fact that $Z_y^{h_s}$ are pair-wise independent (over distinct y's), and the second inequality follows from the trivial fact that a 0-1 random variable has variance at most $1/4$.

Hence, for $\epsilon = 2^{-n/3}$, for all but a $\epsilon/4$ fraction of s,

$$| \Pr_{y \in \{0,1\}^n}[y \circ h_s(y) \in E] - \mu[E] | < \epsilon.$$

♣

We now return to our problem of deterministic amplification for RP languages. Suppose $x \in L$, and the witness set is $W_x \subseteq \{0,1\}^n$. Assume $n \geq 9$ and $\mu(\overline{W_x}) \leq 1/2$ is the error probability of a single random try.

THEOREM 5.25 *If $x \in L$ then*

$$\Pr[G_k(y; s_1, \ldots, s_k) \subseteq \overline{W_x}] \leq (\mu(\overline{W_x}))^{2^k} + (k/4 + 2)\epsilon,$$

where $\epsilon = 2^{-n/3}$.

Proof. The intuitive idea is to use the $k\epsilon$ term to take care of the exceptional probability of "bad" hash functions h_{s_i} in k "rounds", and for "good" hash functions, the estimate should be as if it were uniform, with error term $O(\epsilon)$.

For $1 \leq i \leq k$, fixing any $s_1, \ldots, s_{i-1}$, define the set

$$A_i = \{y \mid G_i(y; s_1, \ldots, s_{i-1}) \subseteq \overline{W_x}\}$$

We will apply the Hash Mixing Lemma to the set $E_i = A_i \times A_i$. We say that s_i is bad for E_i (with respect to the fixed $s_1, \ldots, s_{i-1}$) if s_i belongs to the $\epsilon/4$ fraction of exceptional s in the Hash Mixing Lemma. Note that which s_i is bad depends on $s_1, \ldots, s_{i-1}$. However, for any $s_1, \ldots, s_{i-1}$ there can be at most $\epsilon/4$ fraction of s that can be bad, the rest are "good".

We will use the integration notation $\int$ over the probability measures on s and y. Then

$$\begin{aligned} & \Pr[G_k(y; s_1, \ldots, s_k) \subseteq \overline{W_x}] && (5.4) \\ = & \int_{s_1 \text{ bad}} + \int_{s_1 \text{ good}} \int_{s_2 \text{ bad}} + \cdots + \int_{s_1 \text{ good}} \int_{s_2 \text{ good}} \cdots \int_{s_{k-1} \text{ good}} \int_{s_k \text{ bad}} && (5.5) \\ & + \int_{s_1 \text{ good}} \int_{s_2 \text{ good}} \cdots \int_{s_k \text{ good}} \int_{y}. && (5.6) \end{aligned}$$

The first k terms are all $\leq \epsilon/4$. Note that for the i th term, each $\int_{s_1}, \ldots, \int_{s_{i-1}}$ is over a probability measure.

Now we fix $s_1, \ldots, s_k$ where s_i is good on E_i for $s_1, \ldots s_{i-1}$, and consider

$$\int_y = \Pr_y[G_k(y; s_1, \ldots, s_k) \subseteq \overline{W_x}].$$

Write $\beta = \mu(\overline{W_x})$. Then $\beta \le 1/2$. We will prove by induction that

$$\int_y \le \beta^{2^k} + 2\epsilon.$$

Any upper bound on this probability becomes an upper bound of $\int_{s_1 \text{ good}} \int_{s_2 \text{ good}} \cdots \int_{s_k \text{ good}} \int_y$, as each $\int_{s_i}$ is over a probability measure.

For $k = 0$, trivially $\int_y = \beta$. For $k = 1$, by the Hash Mixing Lemma

$$\int_y \le \Pr_{y,z}[G_1(y; s_1) \circ G_1(z; s_1) \subseteq \overline{W_x}] + \epsilon \le \beta^2 + \epsilon.$$

Similarly, for $k = 2$,

$$\int_y \le (\beta^2 + \epsilon)^2 + \epsilon \le \beta^{2^2} + 2\epsilon,$$

since $2\beta^2 + \epsilon \le 1$. For the general G_{k+1}, with $k \ge 2$,

$$\int_y \le (\beta^{2^k} + 2\epsilon)^2 + \epsilon \le \beta^{2^{k+1}} + 2\epsilon.$$

Here we used $n \ge 3$ and therefore $\epsilon \le 1/8$.

This proves that

$$\Pr[G_k(y; s_1, \ldots, s_k) \subseteq \overline{W_x}] \le (\mu(\overline{W_x}))^{2^k} + (k/4 + 2)\epsilon,$$

For $k \ge 3$, a cleaner expression holds:

$$\Pr[G_k(y; s_1, \ldots, s_k) \subseteq \overline{W_x}] \le (\mu(\overline{W_x}))^{2^k} + k\epsilon.$$

♣

In terms of the number of random bits needed to achieve error probability 2^{-k}, Nisan's generator uses $O(n \log k)$ bits.

5.10.3 Leftover Hash Lemma and Impagliazzo-Zuckerman Generator

Take ℓ and k as integer parameters. (A reasonable choice will be $\ell \approx k \approx \sqrt{n}$ to balance out the bounds.) This generator produces $k + 1$ blocks of n-bits each, using a source of $3n + k\ell$ bits. Let $\{h_s\}$ be a family of universal hash functions from $\{0,1\}^n \to \{0,1\}^{n-\ell}$. The generator G is defined from $\{0,1\}^{2n} \times \{0,1\}^n \times (\{0,1\}^\ell)^k \to (\{0,1\}^n)^{k+1}$ as follows:

$$G(s, Y_1, Z_1, \ldots, Z_k) = Y_1 \circ Y_2 \circ \cdots \circ Y_{k+1},$$

where $\circ$ denotes concatenation, $s \in \{0,1\}^{2n}$, $Y_1 \in \{0,1\}^n$, $Z_i \in \{0,1\}^\ell$, and

$$Y_{i+1} = h_s(Y_i) \circ Z_i,$$

for $1 \leq i \leq k$.

The way this should be read intuitively is this: We first take Y_1 as is. When we use Y_1 to test for membership of an RP language, not all of n-bits of randomness in Y_1 is used. So we should use a hash function to "extract" some "leftover" randomness, getting $h_s(Y_1)$. We supply some fresh true random bits Z_1 to make up Y_2. Then we repeat this process at Y_2, until Y_k.

How good is this generator? The technical tool to address this question is the following Leftover Hash Lemma. Before that we need definitions.

For any discrete probability distribution P and Q on a finite set X, the ℓ_1-distance is defined as

$$||P - Q||_1 = \sum_{x \in X} |P_x - Q_x|,$$

where P_x and Q_x denote the probability of $x \in X$ under P and Q respectively. For any event $E \subseteq X$, the difference in probability $|P(E) - Q(E)| = |\sum_{x \in E} P_x - \sum_{x \in E} Q_x| \leq ||P - Q||_1$. In fact separating out those $P_x > Q_x$ from $P_x \leq Q_x$, it is easy to see that

$$||P - Q||_1 = 2 \max_E |P(E) - Q(E)|,$$

i.e., twice the maximum difference in probability. (This quantity $\max_E |P(E) - Q(E)|$ is called the *statistical distance* between P and Q. However, some authors also call $||P - Q||_1$ their statistical distance.)

A distribution P is caled ϵ-uniform if $||P - U||_1 \leq \epsilon$, where U is the uniform distribution on the same set (same support) as P.

Similarly we can define the ℓ_2-distance

$$||P - Q||_2 = \sqrt{\sum_{x \in X} (P_x - Q_x)^2}.$$

The collision probability $c(P)$ is defined as

$$c(P) = \Pr_{x,x' \in X}[x = x'] = \sum_{x \in X} P_x^2,$$

where x and x' are independently taken according to P.

The next lemma is a simple application of the Cauchy-Schwarz inequality.

LEMMA 5.26 *If $c(P) \leq \frac{1+\epsilon^2}{|X|}$, then P on X is ϵ-uniform.*

Proof. By Cauchy-Schwarz,

$$||P - U||_1^2 \leq |X| \sum_{x \in X} (P_x - \frac{1}{|X|})^2 = |X| \left(c(P) - \frac{1}{|X|} \right) \leq \epsilon^2.$$

♣

In the next lemma we want to compute the collision probability of $s \circ h_s(y)$ when we have some restriction of y (such as y is taken from some (non)-witness set of some RP language.)

LEMMA 5.27 *Let s and y be taken uniformly and independently from $\mathcal{S}$ and $Y' \subseteq Y$, and let h_s be a hash function from Y to T. Let P be the probability distribution $s \circ h_s(y)$, and let $X = \mathcal{S} \times T$. Then*

$$c(P) = \frac{1 + \frac{|T|}{|Y'|}}{|X|}.$$

Proof. We simply compute, taking $s, s' \in \mathcal{S}$ and $y, y \in Y'$ uniformly and independently,

$$\begin{aligned} c(P) &= \Pr[(s = s') \wedge (h_s(y) = h_{s'}(y'))] \\ &= \frac{1}{|\mathcal{S}|} \Pr[h_s(y) = h_{s'}(y') | s = s'] \\ &= \frac{1}{|\mathcal{S}|} \left(\Pr[y = y'] + \Pr[y \neq y'] \cdot \Pr[h_s(y) = h_s(y') | y \neq y'] \right) \\ &\leq \frac{1}{|\mathcal{S}|} \left(\frac{1}{|Y'|} + 1 \cdot \frac{1}{|T|} \right) \\ &= \frac{1}{|X|} \cdot \left(1 + \frac{|T|}{|Y'|} \right) \end{aligned}$$

♣

With the above notations, applying the previous two lemmas in sequence,

LEMMA 5.28 (LEFTOVER HASH LEMMA) *The distribution $s \circ h_s(y)$ is ϵ-uniform, where $\epsilon = \sqrt{|T|/|Y'|}$.*

Now we are ready to prove

THEOREM 5.29 *For $x \in L$ so that $\mu(\overline{W_x}) \leq 1/2$,*

$$\Pr[G(s, Y_1, Z_1, \ldots, Z_k) \subseteq \overline{W_x}] \leq (\mu(\overline{W_x}))^{k+1} + 2^{-\ell/2}.$$

Proof. The intuitive idea of the proof is as follows. We first charge $\mu(\overline{W_x})$ for the first $Y_1 \in \overline{W_x}$. Then on the condition that $Y_1 \in \overline{W_x}$, the hashed value $h_s(Y_1)$ with appended

uniform bits Z_2 makes a distribution quite close to uniform, and upon replacing $h_s(Y_1) \circ Z_2$ by n true random bits $\hat{Y_2}$ we only pay a small price, and an inductive argument takes over.

Now we give the proof. Fix ℓ, we induct on k. Let

$$e_k = \Pr[G(s, Y_1, Z_1, \ldots, Z_k) \subseteq \overline{W_x}],$$

where the probability is over all the unform and independent random choices of $s, Y_1, Z_1, \ldots, Z_k$. Clearly $e_0 = \mu(\overline{W_x})$. For $k \geq 1$,

$$e_k = \mu(\overline{W_x}) \Pr[G(s; Y_2, Z_2, \ldots, Z_k) \subseteq \overline{W_x} \mid Y_1 \in \overline{W_x}],$$

where $Y_2 = \langle h_s(Y_1) \circ Z_1 \rangle$. The probability is still over all $s, Y_1, Z_1, \ldots, Z_k$, except now we have the condition that $Y_1 \in \overline{W_x}$.

Thus we can consider $\overline{W_x}$ as our subset Y' in Lemma 5.27, and estimate the statistical distance between the distributions

$$\langle s \circ Y_2 \circ Z_2 \circ \ldots \circ Z_k \rangle$$

and

$$\langle s \circ \hat{Y_2} \circ Z_2 \circ \ldots \circ Z_k \rangle$$

where $\hat{Y_2}$ is independent and uniform over $\{0,1\}^n$, the same length as $Y_2 = \langle h_s(Y_1) \circ Z_1 \rangle$. Let $\epsilon = \sqrt{2^{n-\ell}/(\mu(\overline{W_x})2^n)} = 1/\sqrt{2^\ell \mu(\overline{W_x})}$ as in Leftover Hash Lemma, then

$$\Pr[G(s, Y_2, Z_2, \ldots, Z_k) \subseteq \overline{W_x} \mid Y_1 \in \overline{W_x}] \leq \Pr[G(s, \hat{Y_2}, Z_2, \ldots, Z_k) \subseteq \overline{W_x}] + \epsilon/2,$$

as the probability over any event, such as the behavior under G, under these two distributions can only differ by at most $\epsilon/2$, half the ℓ_1-norm distance.

It follows that,

$$e_k \leq \mu(\overline{W_x})(e_{k-1} + \epsilon/2).$$

We will inductively assume

$$e_{k-1} \leq (\mu(\overline{W_x}))^k + 2^{-(\ell+1)/2},$$

then we have

$$e_k \leq \mu(\overline{W_x}) \left((\mu(\overline{W_x}))^k + 2^{-(\ell+1)/2} + \frac{1}{2\sqrt{2^\ell \mu(\overline{W_x})}} \right) \leq (\mu(\overline{W_x}))^{k+1} + 2^{-(\ell+1)/2}.$$

The last inequality follows from the fact that $\mu(\overline{W_x}) \leq 1/2$, and $\sqrt{\mu(\overline{W_x})/2^\ell} \leq 2^{-(\ell+1)/2}$.

♣

5.10.4 Expander Mixing Lemma

Let $G = (V, E)$ be a d-regular undirected graph with $n = |V|$ nodes. The adjacency matrix of G is a symmetric 0-1 matrix defined as

$$M(i,j) = \begin{cases} 1 & \text{if } \{i,j\} \in E \\ 0 & \text{otherwise.} \end{cases}$$

Any real symmetric matrix has n real eigenvalues $\lambda_0, \lambda_1, \ldots, \lambda_{n-1}$ (counting multiplicity) and n corresponding orthogonal unit eigenvectors $v_0, v_1, \ldots, v_{n-1}$. Thus the inner product $\langle v_i, v_j \rangle = \delta_{ij}$, and $Mv_i = \lambda_i v_i$. It is clear that the all-1 vector is an eigenvector with eigenvalue d. Normalizing we let $v_0 = \frac{1}{\sqrt{n}}(1, 1, \ldots, 1)^T$, and $\lambda_0 = d$. Let

$$\lambda = \max_{1 \le i \le n-1} |\lambda_i|$$

denote the second largest eigenvalue in absolute value. The gap between d and λ controls the expansion property of G.

We can express every vector v in terms of the eigenvectors $v = \sum_{0 \le i \le n-1} \alpha_i v_i$, then $Mv = \sum_{0 \le i \le n-1} \alpha_i \lambda_i v_i$.

For any two sets $A, B \subseteq V$, let $E(A, B)$ denote the set of ordered pairs of vertices which are also edges of G,

$$E(A,B) = \{(a,b) \mid a \in A, b \in B, \{a,b\} \in E\}.$$

What should the size of $E(A, B)$ be approximately? If we fix A, and randomly pick a subset B of a certain cardinality $|B|$, then $|E(A, B)|$ is a sum of random variables $X_1 + \ldots + X_{|B|}$, where each X_j is the number of ordered pairs of the form (a, b_j), such that $\{a, b_j\} \in E$, where $a \in A$ and b_j is the j th element of the set B. These X_j are not independent; but each b_j is still uniformly distributed on V. Each X_j can be further expressed as a 0-1 sum of $|A|$ many 0-1 random variables, $X_j = X_{1,j} + \ldots + X_{|A|,j}$, where each $X_{i,j}$ is the contribution of the i th element a_i of A to X_j. Clearly then, the expectation $\mathbf{E}[X_{i,j}] = d/n$, and $|E(A, B)|$ has expectation $d|A||B|/n$.

The following lemma says $|E(A, B)|$ is close to this average, the extent to which this is close is controlled by λ.

Lemma 5.30 (Expander Mixing Lemma) *For all $A, B \subseteq V$,*

$$\left| |E(A,B)| - \frac{d|A||B|}{n} \right| \le \lambda \sqrt{|A||B|} \le \lambda n.$$

Proof. Express the charecteristic functions χ_A and χ_B in terms of the eigenvectors,

$$\chi_A = \sum_{0 \le i \le n-1} \alpha_i v_i, \quad \text{and} \quad \chi_B = \sum_{0 \le i \le n-1} \beta_i v_i,$$

where $\alpha_i = \langle \chi_A, v_i \rangle$ and $\beta_i = \langle \chi_B, v_i \rangle$. Clearly $\alpha_0 = |A|/n$ and $\beta_0 = |B|/n$.

Our proof tries to separate out the "main term" and the rest in the sum

$$\begin{aligned}
|E(A,B)| &= \sum_{i \in A, j \in B} M(i,j) \\
&= \chi_A^T M \chi_B \\
&= \left(\sum_{0 \le i \le n-1} \alpha_i v_i \right) \cdot \left(\sum_{0 \le i \le n-1} \beta_i \lambda_i v_i \right) \\
&= \sum_{0 \le i \le n-1} \lambda_i \alpha_i \beta_i \\
&= \frac{d|A||B|}{n} + \sum_{1 \le i \le n-1} \lambda_i \alpha_i \beta_i
\end{aligned}$$

Thus,

$$\begin{aligned}
\left| |E(A,B)| - \frac{d|A||B|}{n} \right| &\le \lambda \sum_{1 \le i \le n-1} |\alpha_i \beta_i| \\
&\le \lambda ||\alpha||_2 ||\beta||_2 \\
&= \lambda ||\chi_A||_2 ||\chi_B||_2 \\
&= \lambda \sqrt{|A||B|} \\
&\le \lambda n
\end{aligned}$$

♣

There is an alternative way to state this result that brings home the similarity to the Hash Mixing Lemma. Consider either (1) taking a uniform $x \in V$ and then taking a uniform neighbor $e_s(x)$ (where s picks one of d values uniformly), or (2) taking two independent and uniform points x and y in V. Then for all $A, B \subseteq V$,

$$|\Pr_{x,s}[x \in A, e_s(x) \in B] - \Pr_{x,y}[x \in A, y \in B]| \le \lambda/d.$$

Note that the first term is $\frac{|E(A,B)|}{nd}$, and the second term is $\frac{|A||B|}{n^2}$, then

$$\left| \frac{|E(A,B)|}{nd} - \frac{|A||B|}{n^2} \right| = \frac{1}{nd} \left| |E(A,B)| - \frac{d|A||B|}{n} \right| \le \frac{\lambda}{d}.$$

Expander graphs are those with a large gap between λ and $d = \lambda_0$. One can explicitly construct expander graph families with $n \to \infty$, and with constant d, and $\lambda \le 2\sqrt{d}$.

For all n' and d', one can construct expander graphs with $n' \le n \le 2n'$ and $d' \le d \le 2d'$, and achieves $\lambda \le d^{9/10}$.

A simple application is the following Karp-Pippenger-Sipser generator. Consider an expander graph with vertex set $\{0,1\}^n$, and degree d. Assume $\lambda \leq d^{9/10}$. With "seed" $z \in \{0,1\}^n$, $G(z)$ produces all d neighbors of z: $G(z) = \{z_1, \ldots, z_d\}$.

THEOREM 5.31 *For $x \in L$ so that $\mu(\overline{W_x}) \leq 1/2$,*

$$\Pr[G(z) \subseteq \overline{W_x}] \leq 2\left(\frac{\lambda}{d}\right)^2.$$

Proof. Let A be the set of vertices all of whose neighbors are in $\overline{W_x}$. Let $B = W_x$. Then $E(A,B) = \emptyset$. By Expander Mixing Lemma,

$$\left|0 - \frac{d|A||W_x|}{2^n}\right| \leq \lambda\sqrt{|A||W_x|}.$$

Since $|W_x| \geq 2^n/2$, we get

$$\frac{|A|}{2^n} \leq 2\left(\frac{\lambda}{d}\right)^2.$$

♣

5.10.5 Ajtai-Komlós-Szemerédi Generator

The idea of using expander graphs can be significantly furthered. The intuitive idea is that, after the initial random vertex, one can take a random walk on the expander graph. If the graph has vertex set $\{0,1\}^n$, and constant degree d, then each additional step of the random walk takes only $O(1)$ random bits. So if we take k steps, we will use only $n+O(k)$ bits. In the meanwhile the names of the vertices the random walk visited form k blocks of n-bits each. Conditional on the previous vertices being in (or not in) the witness set $W_x \subseteq \{0,1\}^n$, the properties of the expander graph ensure that the random walk tends to "disperse" quickly, and therefore these names of the vertices on the random walk in a way simulate true random samples.

Consider an expander graph with vertex set $\{0,1\}^n$, and constant degree d. We will assume $\lambda/d \leq 1/4$; this can be constructed. Take the seed bits $z \in \{0,1\}^n$, and $s_1, s_2, \ldots, s_k \in \{1, \ldots, d\}$. We take $y_0 = z$, and for $i \geq 1$, y_i is the neighbor of y_{i-1} indexed by s_i. Then

$$G(z; s_1, s_2, \ldots, s_k) = \{y_0, y_1, \ldots, y_k\}.$$

THEOREM 5.32 *For $x \in L$ so that $\mu(\overline{W_x}) \leq 1/2$,*

$$\Pr[G(z; s_1, s_2, \ldots, s_k) \subseteq \overline{W_x}] \leq 2^{\Theta(k)}.$$

Proof. The proof idea is to express the probability distributions after i steps of the random walk in the spectra (eigenvalues and eigenvectors) of the expander graph, and analyze the probability (expressed in ℓ_1-norm) in terms of ℓ_2-norm.

Write $N = 2^n$. Choosing uniformly $z \in \{0,1\}^n$ gives the uniform distribution, which can be represented by a vector in $\mathbf{R}^N$ as $p_0 = (1/N, \ldots, 1/N)^T$. Let $\hat{M} = \frac{1}{N}M$ be the probability transition matrix of the Markov chain defined by the random walk, where M is the adjacency matrix of G. If p is a distribution on G, then after one step of the random walk, the distribution is $\hat{M}p$. Define $P = P_{\overline{W_x}}$ to be the projection matrix, which has a 1 in every diagonal entry corresponding to an element of $\overline{W_x}$, and 0 elsewhere else. Then

$$\Pr[y_1 \in \overline{W_x}] = ||P\hat{M}p_0||_1.$$

Denote by $w_i = \Pr[\{y_1, \ldots, y_i\} \subseteq \overline{W_x}]$. We show by induction that

$$w_k = ||(P\hat{M})^k p_0||_1.$$

Inductive Hypothesis: We assume the distribution of the i th step y_i, conditional on $\{y_1, \ldots, y_{i-1}\} \subseteq \overline{W_x}$, is given by the vector

$$\frac{1}{w_{i-1}}\hat{M}(P\hat{M})^{i-1}p_0.$$

and

$$w_{i-1} = ||(P\hat{M})^{i-1}p_0||_1.$$

Then clearly

$$\Pr[\{y_i \in \overline{W_x} \mid \{y_1, \ldots, y_{i-1}\} \subseteq \overline{W_x}] = \frac{1}{w_{i-1}}||(P\hat{M})^i p_0||_1,$$

and

$$\begin{aligned} w_i &= \Pr[\{y_1, \ldots, y_i\} \subseteq \overline{W_x}] \\ &= \Pr[\{y_1, \ldots, y_{i-1}\} \subseteq \overline{W_x}] \cdot \Pr[\{y_i \in \overline{W_x} \mid \{y_1, \ldots, y_{i-1}\} \subseteq \overline{W_x}] \\ &= w_{i-1} \cdot \frac{1}{w_{i-1}}||(P\hat{M})^i p_0||_1 \\ &= ||(P\hat{M})^i p_0||_1. \end{aligned}$$

We can also write

$$\Pr[\{y_i \in \overline{W_x} \mid \{y_1, \ldots, y_{i-1}\} \subseteq \overline{W_x}] = \frac{w_i}{w_{i-1}}.$$

The distribution of y_{i+1}, conditional on $\{y_1, \ldots, y_i\} \subseteq \overline{W_x}$, can be represented by the vector

$$\frac{1}{w_i/w_{i-1}}\hat{M}P\left(\frac{1}{w_{i-1}}\hat{M}(P\hat{M})^{i-1}p_0\right) = \frac{1}{w_i}\hat{M}(P\hat{M})^i p_0.$$

This completes the induction.

(We can also show $\Pr[\{y_0, y_1, \ldots, y_k\} \subseteq \overline{W_x}] = ||(P\hat{M})^k \hat{M} p_0||_1$. But this only contributes another factor of 1/2.)

Now write any vector v in terms of the eigenvectors v_i, $v = \sum_{0 \le i \le n-1} \alpha_i v_i$, then

$$\begin{aligned} ||P\hat{M}v||_2 &\le ||P\hat{M}\alpha_0 v_0||_2 + ||P\hat{M} \sum_{1 \le i \le n-1} \alpha_i v_i||_2 \\ &\le ||P\alpha_0 v_0||_2 + ||\hat{M} \sum_{1 \le i \le n-1} \alpha_i v_i||_2. \end{aligned}$$

Here we used the fact that $\hat{M}v_0 = v_0$, and the orthogonal projection matrix P has ℓ_2-norm $||P|| \le 1$. More concretely, for any vector u, $||Pu||_2^2 = \sum_{i \in \overline{W_x}} u_i^2 \le \sum_{1 \le i \le N} u_i^2 = ||u||_2^2$.

Recall $v_0 = \frac{1}{N}(1, \ldots, 1)^T$. Note that $||Pv_0||_2^2 = \frac{|\overline{W_x}|}{N} = \mu(\overline{W_x})$, and $\hat{M}v_i = (\lambda_i/d)v_i$, it follows that

$$\begin{aligned} ||P\hat{M}v||_2 &\le \sqrt{\mu(\overline{W_x})}|\alpha_0| + ||\sum_{1 \le i \le n-1} (\lambda_i/d)\alpha_i v_i||_2 \\ &= \sqrt{\mu(\overline{W_x})}|\alpha_0| + \sqrt{\sum_{1 \le i \le n-1} (\lambda_i/d)^2 |\alpha_i|^2} \\ &\le \sqrt{\mu(\overline{W_x})}||\alpha_0 v_0||_2 + (\lambda/d)||\sum_{1 \le i \le n-1} \alpha_i v_i||_2 \\ &\le \left(\sqrt{\mu(\overline{W_x})} + \frac{\lambda}{d}\right) ||v||_2, \end{aligned}$$

where the last inequality is because both $\alpha_0 v_0$ and $\sum_{1 \le i \le n-1} \alpha_i v_i$ are orthogonal projections of $v = \sum_{0 \le i \le n-1} \alpha_i v_i$.

Finally, by Cauchy-Schwarz,

$$\begin{aligned} w_k &= \Pr[\{y_1, \ldots, y_k\} \subseteq \overline{W_x}] \\ &= ||(P\hat{M})^k p_0||_1 \\ &\le \sqrt{N}||(P\hat{M})^k p_0||_2 \\ &\le \sqrt{N}\left(\sqrt{\mu(\overline{W_x})} + \frac{\lambda}{d}\right)^k ||p_0||_2 \\ &\le \sqrt{N}\left(\sqrt{1/2} + 1/4\right)^k \frac{1}{N} \\ &= 2^{-\Theta(k)} \end{aligned}$$

In order to achieve error probability 2^{-k}, the AKS generator uses $n + O(k)$ true random bits.