Randomized Methods in Computation – Lecture Notes

Oded Goldreich
Department of Computer Science and Applied Mathematics
Weizmann Institute of Science, Israel.
Email: oded@wisdom.weizmann.ac.il

Spring 2001

Preface

A variety of randomized methods are being employed in the study of computation. The aim of the current course is to make the students familiar with some of these methods. We wish to stress three aspects regarding this course:

1. This course focuses on *methods* (i.e., tools and techniques) and not on concepts.

2. These methods are *randomized* and so the result is often not "(fully) explicit".

3. The focus is on *applications to the study of computation.*

Specific topics included:

- Elements of Probability Theory: Linearity of expectation, Markov Inequality, Chebyshev's Inequality, and Laws of Large Numbers (for pairwise-independent, k-wise independent, and fully independent samples).

- The *Probabilistic Method*: Applications to Max3SAT and the existence of (3-regular) expander graphs.

- Pairwise-independence, hashing, and related topics: Constructions (via matrices and low-degree polynomials), and applications (to Max3SAT, Approximate Counting, and Uniform Generation).

- Small bias spaces, the XOR Lemma, application to hardness of MaxQE(2).

- Expander graphs and random walks on them: Eigenvalues versus expansion, analysis of random walks (Hitter), and the Median-of-Average Sampler.

- Some randomized algorithms: Randomized Rounding (applied to Max3SAT), and Primality Testing (via SQRT extraction).

The lecture notes were taken by students attending the course, which was given in the Spring 2001 semester at the Weizmann Institute of Science. The background and level of the students attending the class was quite mixed, and this forced a slower pace than originally planned.

Original Plan

1. Elements of Probability Theory and the *Probabilistic Method* (2-3 lectures) :
    - Linearity of expectation. Example: for every 3CNF there exists a truth assignment that satisfies 7/8 of the clauses.
    - Markov Inequality and its underlying reasoning. Application: finding a truth assignment as above in probabilistic polynomial-time.
    - Chebyshev's Inequality and its application to the sum/average of pairwise-independent random variables. Extension to k-wise independent sampling (with a proof).
    - Chernoff/Hoeffding Bound (with a proof sketch).
    - Using the Probabilistic Method to prove the existence of 3-regular expanders.
    - Using the Probabilistic Method to prove the existence of (non-trivial) pseudorandom distributions.
    - Lovasz Local Lemma.
2. Pairwise-independence, hashing, and applications (2-3 lectures):
    - Constructions: Using linear transformations (by arbitrary or Toeplitz matrices), and using low-degree polynomials (to obtain k-wise independence). Discussion of the technical difficulties regarding the finite field construction.
    - Application to derandomizing the above Max3SAT algorithm.
    - The ("Leftover") Hashing Lemma.
    - Approximate Counting and Uniform Generation of NP-witnesses with an NP-oracle.
    - Hitters and Samplers (first encounter).
3. Small bias spaces, codes, and applications (1-2 lectures):
    - The distance from uniform distribution over n-bit strings versus the bias of the XOR of some of the bits.
    - A construction of small bias sample spaces.
    - Application: NP-hardness of MaxSAT for Quadratic Equations over GF(2).
    - Relation to linear error correcting codes.
4. Expander graphs and random walks on them (2-3 lectures):
    - Eigenvalues versus expansion.
    - Hitter: Analysis of a random walk on an expander.
    - The Median-of-Averages Sampler.
5. Randomness-Extractors (1-2 lectures): constructions and applications.
6. Some randomized algorithms (2-3 lectures):
    - Finding perfect matchings.
    - Randomized Rounding (applied to MaxSAT).
    - Primality Testing via Modular Square Root extraction.

State and usage of these notes

These notes are neither complete nor fully proofread, let alone being far from uniformly well-written (although the notes of some lectures are quite good). Furthermore, due to the mixed level of the class, the pace was slower than I originally planned, and so these notes include less material than originally intended. Still, I do believe that these notes suggest a good outline for a course on the subject.

Bibliographic Notes

There are several books and lecture notes that cover parts of the material. These include:

- N. Alon and J.H. Spencer: *The Probabilistic Method*, John Wiley & Sons, Inc., 1992.
- O. Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness.* Algorithms and Combinatorics series (Vol. 17), *Springer*, 1998.

  Appendix B has been revised and appears as a separated text called "A Taste of Randomized Computations", available from `http://www.wisdom.weizmann.ac.il/~oded/rnd.html`.
- O. Goldreich: *Introduction to Complexity Theory – Lecture Notes*, 1999. Available from `http://www.wisdom.weizmann.ac.il/~oded/cc.html`.
- R. Motwani and P. Raghavan: *Randomized Algorithms*, Cambridge University Press, 1995.

However, the presentation of the material in the current lecture notes does not necessarily follow these sources.

In addition to the above general sources, we refer the reader to bibliographic notes presented at the end of each lecture.

Acknowledgments

I am most grateful to the students who have attended the course and partipiated in the project of preparing the lecture notes. So thanks to *Omer Angel, Boaz Barak, Uri Bernholz, Eden Chlamtac, Ariel Elbaz, Lev Faivishevsky, Maksim Frenkel, Safro Ilya, Amos Gilboa, Ya'ara Goldschmidt, Olga Grinchtein, Dani Halevi, Yehuda Hassin, Eran Keydar, Sergey Khristo, Shimon Kogan, Amos Korman, Oded Lachish, David Lehmann, Yehuda Lindell, Itsik Mantin, Yaniv Meoded, Shai Mor, Hani Neuvirth, Lior Noy, Eran Ofek, Yoav Rodeh, Alon Rosen, Yaki Setty, Ido Shaked, Denis Simakov, Benny Stein, Eran Tromer, Erez Waisbard,* and *Udi Wieder.*

I am grateful to *Shafi Goldwasser* who gave a guest lecture (on Primality Testing via SQRT extraction) during the course.

Lecture Summaries

Lecture 1: Probability and the Probabilistic Method. After a general introduction to the course, we recall some of the basic notions of probability theory, such as the expectation and the variance of a random variable, Chebyshev's inequality and the first law of large numbers. We also introduce the Probabilistic Method and demonstrate one application of it (specifically, to the Max3SAT problem). Finally, we develop a simple algorithm that given a 3CNF formula, finds an assignment that satisfies many of its clauses.

Notes taken by Maksim Frenkel and Safro Ilya.

Lecture 2: Laws of Large Numbers and Existence of Expanders. We focus on various laws of large numbers. These laws bound the probability that a random variable deviates far from its expectation. Motivated by the problem of approximating the average of a function, we present stronger bounds than Chebyshev's inequality: moving from pairwise independent sampling to $2k$-wise independent sampling and then to totally independent sampling makes the the error probability exponentially vanishing (in the amount of independence). We also define families of graphs called expanders, and prove the existence of a family of 3-regular expanders (using the probabilistic method).

Notes taken by Shai Mor and Erez Waisbard.

Lecture 3: Small Pairwise-Independent Sample Spaces. Often in applications that use randomness (i.e., coin tosses) we do not need all these tosses to be independent, instead it may suffice that they are k-independent for some k. In the case of totally independent random variables the probability space is at least of the size $|S|^m$, where m is the number of variables in the sequence and S is the sample space for an individual variable. In contrast, we present a construction for pairwise independent sample space of the size $\max(|S|, m)^2$ and its extension to the k-wise independent case. This will allow us to *derandomize* the algorithm (presented on the first lecture) that finds assignments for 3CNF that satisfies at least 7/8 clauses. The resulting algorithm will conduct an exhaustive search of the relatively small 3-wise independent sample space (of boolean assignments).

Notes taken by Lev Faivishevsky and Sergey Khristo.

Lecture 4: Small Pairwise-Independent Sample Spaces (cont.) and Hash Functions. A good construction of k-wise independent sample space should be small and convenient to use in applications. The construction presented in the previous lecture satisfies the first requirement (i.e., is small), but is rather complex, imposing a number of technical problems in application. In this lecture a second, much more convenient construction is presented, based on affine transformations. This construction yields small sample spaces, but only for pairwise independence ($k = 2$). We also start discussing hash functions, showing their relation to small pairwise-independent sample spaces, and presenting a Hashing Lemma. The latter asserts that pairwise-independent hash functions map sets in a "smooth" manner (i.e., each image obtains about the same number of preimages).

Notes taken by Olga Grinchtein and Denis Simakov.

Lecture 5: Approximate Counting. We discuss quantitative problems related to NP-relations. Specifically, given an element in an NP-language, we ask how many witnesses this element has. We use Hash functions in order to present an efficient randomized procedure that given an an instance in an NP-language and oracle access to NP, approximates the number of witnesses for that instance.

Notes taken by Oded Lachish, Eran Ofek and Udi Wieder.

Lecture 6: Uniform Generation. This lecture continues the discussion of approximate counting and uniform generation using an NP-oracle begun in the previous lecture, where we saw an approximate counting algorithm. Here we will see the equivalence (up to Turing reductions) of approximate counting to uniform generation of NP-witnesses. Together with the approximate counting algorithm from last lecture, this yields a uniform generation algorithm using an NP-oracle. We conclude with an alternative (direct) uniform generation algorithm, which uses n-wise independent hash functions (as opposed to the pairwise independent hash functions used for approximate counting).

Notes taken by Eden Chlamtac and Shimon Kogan.

Lecture 7: Small Bias Sample Spaces (Part 1). In this lecture we introduce the notion of ϵ-bias sample spaces. Informally, a random variable over $\{0,1\}^n$ is ϵ-bias if for every subset of bits, the difference between the probability that the parity of the bits is zero and the probability it is one is at most ϵ. We show a connection between ϵ-bias and the statistical distance to the uniform distribution. Next, we present an application of ϵ-bias sample spaces for generating "approximately" k-wise independent distributions (i.e., distributions over $\{0,1\}^n$ in which every k bits look almost uniformly distributed over $\{0,1\}^n$).

Notes taken by Yehuda Lindell and Alon Rosen.

Lecture 8: Small Bias Sample Spaces (Part 2) In this lecture we present a construction of small bias sample spaces; that is, we present an algorithm that given n and ϵ outputs an explicit representation of an ϵ-bias sample space over $\{0,1\}^n$ and does so within time poly(n/ϵ). We also present an application of such a construction to proving a tight result regarding the hardness of approximating MaxQE (where one is given a sequence of quadratic equations over $GF(2)$ and is asked to find an assignment that satisfies as many as possible of these equations).

Notes taken by Boaz Barak and Itsik Mantin.

Lecture 9: Expansion and Eigenvalues. This lecture is about expander graphs, which are very useful in many probabilistic applications. First, we would describe some of the combinatorial definitions and properties of an expander graph. Second, we would see one application which uses expander graphs, and finally, we would discuss some of the algebraic properties of such graphs.

Notes taken by Ya'ara Goldschmidt, Eran Keydar and Yaki Setty.

Lecture 10: Random Walks on Expanders. In this lecture we show that taking an l-step random walk in an expander graph is in a way similar to choosing l vertices at random. The advantage of using a random walk over taking an independent sample is that a random walk on a d-regular graph requires $\log_2 d$ random bits per each step, whereas selecting each random vertex of an N-vertex graph requires $\log_2 N$ random bits. As a warm up, we show that satring at any vertex in an N-vertex expander and taking $O(\log N)$ random steps, one reaches approximately the uniform distribution (on the vertices). We also recall the relation between two definitions of expanders and describe some known constructions of expanders.

Notes taken by Omer Angel, Dani Halevi and Amos Gilboa.

Lecture 11: Primality Testing (via SQRT extraction). In this lecture we present two randomized algorithms in number theory. The first algorithm finds in expected polynomial-time a square root of a number in Z_p^* for any prime number p. We then use this algorithm to construct a polynomial time algorithm with two-sided errors for testing primality.

Notes taken by Yehuda Hassin, Lior Noy and Benny Stein.

Lecture 12: Hitters and Samplers. A *sampler* is an oracle Turing machine that estimates the average of any function $f : \{0,1\}^n \to [0,1]$ with bounded deviation and bounded probability of failure. A *hitter* is an oracle Turing machine that, given a boolean function $f : \{0,1\}^n \to \{0,1\}$, finds x s.t. $f(x) = 1$ with a bounded probability of failure if f has value 1 on at least a constant fraction of $\{0,1\}^n$. These are fundamental and useful problems. We consider the randomness and query complexities of hitters and samplers whose running time is polynomially bounded, and recall lower bounds for each. We show several simple constructions for both, as well as improved constructions based on pairwise-independent sample spaces and on random walks on expander graphs. We then show composite constructions whose complexities match the lower bounds.

Notes taken by Amos Korman, Yoav Rodeh and Eran Tromer.

Lecture 13: Randomized Rounding. Some approximation algorithms use linear programming on a variant of the original problem. The solutions of a linear programming problem are non-integral, in the general case, while legitimate solutions to the original problems are integers only. One idea which can be used to turn real numbers into integer solutions, is to use the linear programming (non-integer) solutions as probability parameters for selecting (integer) solutions to the original problem. In this lecture we present this idea and demonstrate it in two cases.

Notes taken by Ariel Elbaz, David Lehmann and Ido Shaked.

Contents

Lecture 1

Probability and the Probabilistic Method

Notes taken by Maksim Frenkel and Safro Ilya

Summary: This lecture is a general introduction to the course "Randomized methods in Computation". We review some of the basic notions of probability theory, such as the expectation and the variance of a random variable, Chebyshev's inequality and the first law of large numbers. We also introduce the Probabilistic Method and demonstrate one application of it (specifically, to the Max3SAT problem). Finally, we will develop a simple algorithm that given a 3CNF formula, finds an assignment that satisfies many of its clauses.

1.1 Motivation

Oded's Note: *I tried to stress three aspects regarding this course as hinted by its name: First, this course focuses on* methods *(i.e., tools and techniques) and not on concepts. Second, these methods are* randomized *and so the result is often not "(fully) explicit". This can be seen both in the form of randomized algorithms (and procedures) that do not always return the same answer (but rather have an output distribution that satisfies some desired property with high probability) as well as in arguments (e.g., in the "probabilistic method") that do not explicitly present a desired object (but rather only specifies a distribution of objects so that with high probability an object in this distribution possesses the desired property). Lastly, the third aspect of this course is that it focuses on* applications to the study of computation. *This is reflected in the emphasis put on the complexity of generating certain distributions, where complexity may mean a variety of complexity measures (such as time, space, and randomness).*

Consider the following problem: given a set of integer numbers, sort them in an ascending order. There exist a lot of different algorithms that solve this problem. These algorithms have different worst and average running times. Some of them are kno wn to perform badly on certain problem instances (for example the QuickSort algorithm) while on the average their running times are good. A possible solution is to randomly permute the given integer sequence such that it will resemble the average case inp ut. In this example our algorithm will use the sequence of outcomes of coin tosses which explicitly determine the permutation. Such an algorithm is called *randomized*.

A randomized algorithm gets as its input a sequence of n outcomes of coin toss in addition to its regular input. The algorithm makes its decisions based upon this sequence. Standard (deterministic) algorithm has one answer given one input instance. Randomized algorithm has a set of answers (or a set of ways to reach an answer) given the problem input instance. Some of these answers (or ways to reach an answer) are good whenever other are bad. The advantage of a good randomized algorithm is its abil ity to reach the good answer (or to find a good way to this answer) fast.

> Oded's Note: *The above motivation (which is not at all what I said in class) addresses just one aspect of the course. See above note for what I did say.*

1.2 Basic Notions from Probability Theory

> Oded's Note: *I assume that most students know most of the contents of the current section. My goals in covering this basic material were two-fold. First, to notify all student of what they are assumed to know, and teach the few that don't know some of this material the little they don't know. Second, in a confusing business such as probabilistic analysis it is good to get reminded of the basic underlying definitions: in case of difficulty one should remember that the complex notions can and should (when in doubt) be traced back to the underlying notions. That is, when faced with a complex analysis it is a good practice to ask basic questions like "what is the underlying probability space" (which is typically only implicit in complex discussions).*

1.2.1 Outcomes, Events and Probability

We assume that the reader is familiar with the basic notions of Probability Theory. Therefore the purpose of this section will be to review those concepts, which will be useful for us in upcoming lectures.

Let us consider an experiment that results in several outcomes. Denote these outcomes by $A_1, A_2, ..., A_m$. If we will repeat this experiment n times under various conditions, A_1 will occur k_1 times, while other outcomes will occur $n - k_1$ times. We refer to a certain subset of the outcomes as *an event* and to the set of all possible outcomes as *a probability space* Ω. When we talk about *the probability* of the event X occurring over its probability space Ω we consider the ratio of the sizes of the two sets: $\frac{|X|}{|\Omega|}$. The meaning of this ratio is the answer to the question: how to measure the possibility that the event X will occur as a result of our experiment. This ratio we denote by $Prob(X)$. It is easy to see that

$$0 \leq Prob(X) \leq 1$$

Example 1.1 *Suppose we are rolling a die and we are interested in obtaining an even outcome. Here*

$$\Omega = \{1, 2, 3, 4, 5, 6\}$$

The event X will be the following subset of Ω:

$$X = \{2, 4, 6\}$$

Then

$$Prob(X) = \frac{|X|}{|\Omega|} = \frac{3}{6}$$

Remark 1.2.1 *In this course we will consider the probability space to be a finite set.*

Sometimes the outcome of one event may influence the outcome of another one. To measure the amount of influence we introduce the *conditional probability* of event B occurring given that the event A has occurred:

$$Prob(B|A) = \frac{Prob(A \cap B)}{Prob(A)}$$

Recall that events can be either *independent* or *dependent*. The events are independent when the trials of our experiment are carried out independently, which implies that the result of each trial does not influence the other results. If we are interested in the probability of the events A and B occurring when they are independent then

$$Prob(A \cap B) = Prob(A) \cdot Prob(B)$$

In other words, A and B are independent if the conditional probability of B given A is just equal to the probability that B occurs without paying attention to A at all, i.e.

$$Prob(B|A) = Prob(B)$$

> Oded's Note: *Indeed, probabilities are merely ratios of sizes of sets, and so probabilistic analysis is merely combinatorics. Yet, as is often the case in mathematics and science, using certain definitions and notations (rather than others) may simplify the analysis tremendously. Specifically, in many cases, the analysis is much simpler to carry out in terms of probabilities than in terms of set sizes.*

1.2.2 Random variables

A random variable is a function from a probability space to the reals (or complexes); that is, a random variable X is

$$X : \Omega \longrightarrow \mathsf{R} \qquad (possibly\ \mathsf{C})$$

As we remarked earlier, Ω in our case is a finite set and hence the set of values that X is mapped onto is also finite. Intuitively a random variable is a function that can be calculated as a result of an experiment. The adjective "random" reminds us that we are dealing with a sample space and are trying to describe "random events". Random variables are often defined without explicitly specifying a probability space (but such a space is always implicit in the definition). For example, the number of hits out of three shots, the number of tails in five coin tosses, etc. Once we have defined what our random variable measures we can define the probability that it will take on a certain value α:

$$Prob[X = \alpha] = \frac{|\{w \in \Omega : X(w) = \alpha\}|}{|\Omega|}$$

Let us remind also that if X and Y are random variables then

$$X + Y,\ \ X - Y,\ \ XY,\ \ \frac{X}{Y},\ \ \alpha X + \beta Y$$

(where α and β are real numbers) are also random variables.

Discrete random variables X and Y are independent if

$$Prob(X = x\ and\ Y = y) \ = \ Prob(X = x) \cdot Prob(Y = y)$$

for all values x in the range of X and y in the range of Y.

By definition of a random variable one can not define it without specifying the corresponding probability space. However, often, when the associated probability space is obvious we will drop this convention. We will usually describe the set of values that can be associated with the random variable with their corresponding weights, instead of explicitly defining the function with its domain.

> Oded's Note: *Indeed the last paragraph refers to my main motivation in recalling all these trivialities. We will often talk of random variables without specifying the underlying probability space, and will often be able to carry out the analysis that way. However, when in doubt, one should specify explicitly the probability space that underliees the analized events or random variables.*

1.2.3 Expectation

For a random variable X defined on a probability space Ω, its *expectation* is the number $E(X)$ defined by the formula

$$E(X) = \sum_{\omega \in \Omega} Prob(\{\omega\}) X(\omega)$$

If we think of $Prob(\{\omega\})$ as the weight attached to ω then $E(X)$ can be understood as the weighted average of the function X. The intuition of the value $E(X)$ might be obtained from its analogue in mechanics. Consider a line with n physical points distributed on it. Every point has a weight p_i. Let X_i be the location of the i-th point. Taking the sum of all the weights to be 1, the equilibrium point of the system has the coordinate:

$$X = \frac{\sum_{i=1}^{n} p_i X_i}{\sum_{i=1}^{n} p_i} = \frac{\sum_{i=1}^{n} p_i X_i}{1} = \sum_{i=1}^{n} p_i X_i$$

This is exactly the expectation of the random variable X. The following theorem introduces some properties of the expectation. These properties will be very useful in the future discussion.

Proposition 1.2.2 *For two random variables X and Y,*

1. *$E(aX) = aE(X)$, for any real a.*
2. *$E(X+Y) = E(X) + E(Y)$, even if X and Y are dependent.*
3. *If X and Y are independent then $E(XY)=E(X)E(Y)$.*

Proof: Let X be the random variable that takes on discrete values and let a be a number. Let aX be defined on the set Ω, then

$$\begin{aligned} E(aX) &= \sum_v v \cdot Prob(aX = v) \\ &= a \cdot \sum_v \frac{v}{a} \cdot Prob(X = \frac{v}{a}) \\ &= a \cdot E(X) \end{aligned}$$

In order to prove Part 2 let us assume that Y takes on discrete values.

$$
\begin{aligned}
E(X+Y) &= \sum_{(x,y)} [(x+y)\cdot Prob(X=x \text{ and } Y=y)] \\
&= \sum_x \sum_y x \cdot Prob(X=x \text{ and } Y=y) \ + \ \sum_y \sum_x y \cdot Prob(X=x \text{ and } Y=y) \\
&= \sum_x x \cdot Prob(X=x) \ + \ \sum_y y \cdot Prob(Y=y) \\
&= E(X)+E(Y)
\end{aligned}
$$

And finally, for independent X and Y, we have

$$
\begin{aligned}
E(XY) &= \sum_x \sum_y xy \cdot Prob(X=x \text{ and } Y=y) \\
&= \sum_x \sum_y xy \cdot Prob(X=x)\cdot Prob(Y=y) \\
&= \left(\sum_x x \cdot Prob(X=x)\right)\cdot\left(\sum_y y \cdot Prob(Y=y)\right) \\
&= E(X)\cdot E(Y)
\end{aligned}
$$

■

In general, if $X_1, ...X_n$ are random variables and $X = \alpha_1 X_1 \ + \ ... \ + \ \alpha_n X_n$, then

$$E(X) = \alpha_1 E(X_1) \ + \ ... \ + \ \alpha_n E(X_n)$$

The linearity of expectation is a very useful tool in randomized methods. Its power comes from the idea that it may be possible to decompose a complex instance X as the sum of many simple instances X_i, and the linearity will hold regardless of the dependence of X_i's. The expectation of each of the X_i-s can be calculated relatively easy compared to that of X. We will see an application of this principle in Theorem 1.6. Another useful fact regarding the expectation is that it cannot be *always* higher than the random variable. That is:

Proposition 1.2.3 *For any random variable X*

$$Prob[X \geq E(X)] > 0$$

Proof Sketch: We need to show that there exists x, such that $Prob[X=x] > 0$ and $x \geq E(X)$. Let x_{max} be the maximal value that X may be equal to such that $Prob[X = x_{max}] > 0$ and $Prob[X > x_{max}] = 0$.

$$E(X) \ = \ \sum_x Prob[X=x]\cdot x \ \leq \ \sum_{x \leq x_{max}} Prob[X=x]\cdot x_{max} \ = \ x_{max}$$

Therefore $E(X) \leq x_{max}$ and since we know that $Prob[X = x_{max}] > 0$ we have shown that $Prob[X \geq E(X)] > 0$. ■

1.2.4 The Variance

Oded's Note: *The following paragraph is not something I said. I would have said that the variance is a measure of the "spread" or "variability" of a random variable, and that its specific definition is adequate for various Laws of Large Numbers as well as for Chebyshev's Inequality that underlies some of them.*

The variance of a random variable X measures the unpredictability of a random variable or in other words this is a measure of its variation from the mean $E(X)$. Suppose that two random variables X and Y have the same mean μ, but the values of X are distributed more closely to $E(X)$ than are the values of Y to $E(Y)$. That is, the probability is higher that X will be closer to μ than Y will be. We would like to define some measure that will reflect that fact. Let us shift both X and Y to 0 and consider the values $X-\mu$ and $Y-\mu$. Since Y was spread farther around μ than X was, $Y-\mu$ will be spread farther around 0 than $X-\mu$. Let's square the above differences to make sure that they are positive. The *expectation* of $(X-\mu)^2$ will be closer to 0 than $(Y-\mu)^2$. Therefore it makes sense to define the variance of a random variable X as follows:

$$VAR(X) \;=\; E((X-\mu)^2),\ where\ \mu = E(X)$$

Using the linearity of expectation, we see that

$$\begin{aligned} VAR(X) &= E((X-\mu)^2) \\ &= E(X^2 - 2\mu X + \mu^2) \\ &= E(X^2) - 2\mu E(X) + E(\mu^2) \\ &= E(X^2) - 2\mu\mu + \mu^2 \\ &= E(X^2) - E(X)^2 \end{aligned}$$

The above formula provides a *computationally easier* method of evaluating the variance.

The variance is not always linear. An exceptional case when the variance is linear is when applied to a sum of paiwise-independent random variables. Following is a special case (which can be easily generalized to a sequence of pairwise independent random variables):

Proposition 1.2.4 *If X and Y are independent then*

$$VAR(X+Y) \;=\; VAR(X) + VAR(Y)$$

Proof: Since X and Y are independent, $E(XY) = E(X)\cdot E(Y) = \mu_X\mu_Y$, where μ_X and μ_Y equal to $E(X)$ and $E(Y)$ respectively. Then

$$\begin{aligned} VAR(X+Y) &= E((X+Y)^2) - (\mu_X + \mu_Y)^2 \\ &= E(X^2 + 2XY + Y^2) - (\mu_X^2 + 2\mu_X\mu_Y + \mu_X^2) \\ &= E(X^2) - \mu_X^2 + E(Y^2) - \mu_Y^2 + 2E(XY) - 2\mu_X\mu_Y \\ &= VAR(X) + VAR(Y) \end{aligned}$$

since the last two terms cancel. ■

On the other hand, in case X and Y are dependent $VAR(X+Y)$ may be different from $VAR(X)+VAR(Y)$. In particular, if $X=Y$ (i.e., $Prob(X=Y)=1$) then $VAR(X+Y) = 2VAR(X) + 2VAR(Y)$, whereas if $X=-Y$ than $VAR(X+Y)=0$. Both claims follows from the next proposition:

Proposition 1.2.5 *Let a and b be constants. Then*

$$VAR(aX+b) \;=\; a^2 \cdot VAR(X)$$

Proof: Let $\mu = E(X)$ and observe that

$$E(aX+b) \;=\; aE(X)+b \;=\; a\mu+b$$

Now

$$\begin{aligned} VAR(aX+b) &= E((aX+b)^2) - (E(aX+b))^2 \\ &= E(a^2X^2 + 2abX + b^2) - (a\mu+b)^2 \\ &= a^2 \cdot E(X^2) + 2ab \cdot E(X) + b^2 - (a^2\mu^2 + 2ab\mu + b^2) \\ &= a^2 \cdot (E(X^2) - \mu^2) \\ &= a^2 \cdot VAR(X) \end{aligned}$$

■

Note that the above result is intuitive, since the values of the random variable $aX+b$ vary from its expectation $E(aX+b)$ as do the values of aX from its expectation $E(aX)$. On the other hand, simply shifting the distribution by b does not change its *spread*. Another intuitive property of the variance follows:

Proposition 1.2.6 *The variance of a random variable X is zero if and only if X is a constant.*

Proof: The backward direction ($\Leftarrow$) is trivial and follows from Proposition 1.2.5, when $a = 0$. The forward direction follows from the definition of the variance. Specifically, we have

$$VAR(X) \;=\; E((X-\mu)^2) \;=\; 0$$

where $\mu = E(X)$. Since $(X-\mu)^2$ is a nonnegative value, the fact that its expectation is zero implies that $X = \mu$. ■

1.2.5 Markov's and Chebyshev's Inequalities and a Law of Large Numbers

In analyzing the performance of a randomized algorithm we will often need to show that the algorithm behaves well almost all the time. While the concept of the expectation of a random variable provides information regarding the *average case scenario* (for instance, the algorithm's average running time), lots of times we will interested in the "typical" (rather than average) behavior; that is, we want to know an interval of values that is guaranteed to contain the value of the random variable, almost always. In doing so we will need to analyze the probability that a random variable deviates from its expectation by a given amount.

The first inequality we introduce provides a bound on the probability that a random variable deviates from its expectation.

Theorem 1.2 (Markov's Inequality): *If X is a random variable that assumes only non-negative values, then for all real and positive v*

$$Prob[X \geq v] \leq \frac{E(X)}{v}$$

Proof:

$$\begin{aligned} E(X) &= \sum_x Prob[X=x]\cdot x \\ &= \sum_{x\geq v} Prob[X=x]\cdot x \;+\; \sum_{x<v} Prob[X=x]\cdot x \\ &\geq \sum_{x\geq v} Prob[X=x]\cdot v \;+\; \sum_{x<v} Prob[X=x]\cdot 0 \\ &= Prob[X\geq v]\cdot v \end{aligned}$$

Thus

$$Prob[X\geq v] \leq \frac{E(X)}{v}$$

■

An equivalent formulation of Markov inequality is :

$$Prob[X\geq \alpha\cdot E(X)] \leq \frac{1}{\alpha}$$

As we can see the upper bound of the probability that a random variable deviates from its expectation decreases as α increases. As noted before the probability space of X is not directly mentioned in the above theorem but it is obviously implied. Usually we use the Markov Inequality when our knowledge about the distribution is limited to its expectation and at least one bound on the distribution of a random variable. If the variance of a given random variable X is known, the Markov inequality can be used to achieve a better bound on the probability that X will deviate from its expectation, as seen in the next theorem.

Theorem 1.3 (Chebyshev's Inequality): *For any random variable Y and any $\delta > 0$,*

$$Prob[|Y-E(Y)|\geq \delta] \leq \frac{VAR(Y)}{\delta^2}$$

Proof: Let $X=(Y-E(Y))^2$ and note that X is non-negative and that

$$E(X)=E((Y-E(Y))^2)=VAR(Y)$$

Next, observe that

$$Prob[|Y-E(Y)|\geq \delta] = Prob[X\geq \delta^2]$$

Finally, invoking Markov's inequality on X and δ^2 yields

$$Prob[X\geq \delta^2] \leq \frac{E(X)}{\delta^2} = \frac{VAR(Y)}{\delta^2}$$

which completes the proof. ■

Indeed, as expected for *any* distribution, the probability that the values of X are more than a certain *fixed* distance away from its mean gets smaller as $VAR(X)$ gets smaller. A special case where this is useful is for the sum of pairwise independent random variables.

Definition 1.4 *The random variables $X_1,...,X_n$ are called* pairwise independent *if for every $i\neq j$ and all a and b,*

$$Prob[X_i=a\wedge X_j=b] = Prob[X_i=a]\cdot Prob[X_j=b]$$

Corollary 1.2.7 *Let $X_1, ..., X_n$ be pairwise independent random variables with common mean μ and variance σ^2. Denote $S_n = \sum_{i=1}^n X_i$, and let $\overline{X} = S_n/n$. Then*

$$Prob[|\overline{X} - \mu| \geq \epsilon] \leq \frac{\sigma^2}{n\epsilon^2}$$

Proof: First, observe that $VAR(S_n) = n\sigma^2$, by (a generalization of) Proposition 1.2.4.

Oded's Note: *Specifically, we use $VAR(\sum_{i=1}^n X_i) = \sum_{i=1}^n VAR(X_i)$, which is proven as Proposition 1.2.4:*

$$\begin{aligned}
VAR(\sum_{i=1}^n X_i) &= E\left((\sum_{i=1}^n X_i)^2\right) - E\left(\sum_{i=1}^n X_i\right)^2 \\
&= E\left((\sum_{i=1}^n X_i^2) + (\sum_{i \neq j} X_i X_j)\right) - \left(\sum_{i=1}^n E(X_i)\right)^2 \\
&= \sum_{i=1}^n E(X_i^2) + \sum_{i \neq j} E(X_i X_j) - \sum_{i=1}^n E(X_i)^2 - \sum_{i \neq j} E(X_i)E(X_j) \\
&= \sum_{i=1}^n E(X_i^2) - \sum_{i=1}^n E(X_i)^2 \\
&= \sum_{i=1}^n VAR(X_i)
\end{aligned}$$

where the key fact being used is $E(X_i X_j) = E(X_i)E(X_j)$, for $i \neq j$, which follows becuase X_i and X_j are independent.

Now, by Proposition 1.2.5:

$$VAR(\overline{X}) = VAR(S_n/n) = \frac{VAR(S_n)}{n^2} = \frac{\sigma^2}{n}$$

Finally, invoking Chebyshev's inequality on $\overline{X}$, we get

$$Prob[|\overline{X} - \mu| \geq \epsilon] \leq \frac{VAR(\overline{X})}{\epsilon^2} = \frac{\sigma^2}{n\epsilon^2} \tag{1.1}$$

■

Remark 1.2.8 *The above corollary generalizes to the case where the X_i's are pairwise independent, but not necessarily with identical mean. In such a case, we let $\mu = \frac{1}{n}\sum_{i=1}^n E(X_i)$ and have:*

$$Prob[|\overline{X} - \mu| \geq \epsilon] \leq \frac{\max_i VAR(X_i)}{n\epsilon^2}$$

1.3 The Probabilistic Method

1.3.1 Introduction

The probabilistic method is a way to prove the existence of objects having certain properties, by proving their existence "in abundence" (with respect to some probability spcae). That is, suppose we want to prove the existence of some combinatorial object with property T. The construction of such an object may be too hard. The probabilistic method suggests to define some probability space S, and to show that the probability that an object selected uniformly in S satisfies property T with non-zero probability. This would imply the existence of such an object (i.e., satisfying property T).

1.3.2 An application of the probabilistic method: the 3CNF-SAT problem

Definition 1.5 *A 3CNF boolean formula is a boolean formula φ written in conjunctive normal form where each clause contains exactly three* different *literals:*

$$\varphi = \bigwedge_{i=1}^{m} C_i, \qquad \text{where } C_i = \bigvee_{j=1}^{3} l_{ij}$$

each literal l_{ij} can be either a boolean variable or negation of a boolean variable.

Theorem 1.6 *For every 3CNF boolean formula φ there exists a truth assignment that satisfies at least $\frac{7}{8}$ of clauses in φ.*

Note that the theorem is not true in case we consider arbitrary CNF formulae (or CNF formulae in which each clause may have any three literals that are not necessarily different). Consider for example, the formula

$$(x_1 \vee x_2) \wedge (\neg x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_2)$$

Clearly, each possible truth assignment satisfies only 3 out of the 4 possible clauses. A similar example shows that the Theorem 1.6 is indeed optimal: there exists 3CNF boolean formulae for which any truth assignment satisfies only 7/8 fraction of the clauses.

Proof: For sake of simplicity, we assume that no clause of φ may contain two literals corresponding to the same variable (i.e., a boolean variable and its negation). Note that any clause that conatins a boolean variable and its negation is satisfied under any truth assignment. Thus, we may omitting such clauses, observing that any assignment that satisfies 7/8 of the remaining clauses also satisfies all the omitted clauses.

The idea of the proof is to select a random assignment to the variables of φ uniformly (over all possible choices) and to evaluate the expected number of clauses satisfied by it. That is, we take as a probability space the set of all possible assignments to the variables of φ. Choose a random assignment and let X be the number of clauses (of $\varphi = \bigwedge_{i=1}^{m} C_i$) satisfied by it. Let X_i be the indicator random variable such that

$$X_i = \begin{cases} 1 & \text{if the assignment satisfies } C_i \\ 0 & \text{otherwise} \end{cases} \tag{1.2}$$

For $\varphi = \bigwedge_{i=1}^{m} C_i$, we have $X = X_1 + ... + X_m$. These X_i-s are not necessarily independent, because some clauses may contain the same literals (or just the same variables). Nevertheless, this fact does not prevent us from considering the expectation of X and using linearity of expectation:

$$E(X) = E(\sum_{i=1}^{m} X_i) = \sum_{i=1}^{m} E(X_i)$$

Thus we have broken down the relatively complicated problem of calculating the value $E(X)$ into a number of simpler problems of calculating each of the $E(X_i)$-s. Suppose that the clause C_j contains the three literals $l_{j1},\ l_{j2},\ l_{j3}$. Since these three literals are of different variables, there are 8 possible assignments to these three literals and 7 of them satisfy C_j. Since each of these 8 possibilities is equaly likely, when an assignment to all variables (of φ) is selected uniformly, we get:

$$E(X_i) \;=\; Prob[X_i = 1] \;=\; \frac{7}{8}$$

So $E(X) = \frac{7}{8}m$. Now as we already observed (see Proposition 1.2.3) $Prob[X \geq E(X)] > 0$, so we can say that in our case $Prob[X \geq \frac{7}{8}m] > 0$. This implies the existence of a truth assignment for φ that satisfies at least $\frac{7}{8}$ of its clauses. ■

In the above proof we have defined the probability space Ω of all possible truth assignments to a given $3CNF$ formula, and then shown that the probability of choosing an assignment from Ω that satisfies the required property is non-zero. We then concluded that an assignment as required does exist.

1.3.3 Obtaining a randomized algorithm

In the previous section we have shown that there exists an assignment that satisfies at least $\frac{7}{8}$ of the clauses in a given $3CNF$ formula. Here we will present a randomized algorithm that finds such an assignment with high probability in polynomial time. The idea behind this algorithm will be to repeteadly select random assignments to the variables of the given formula, untill it finds an assignment that satisfies at least $\frac{7}{8}$ of the clauses. The expected running time of the algorithm depends on the probability that the chosen random assignment satisfies at least $\frac{7}{8}$ of the clauses. The following claim will show that this probability is quite noticeable (i.e, bigger than a polynomial fraction):

Claim 1.3.1 *Let φ be a 3CNF formula with m clauses. Let X be the random variable that counts the number of clauses satisfied by a uniformly random assignment to the variables of φ. Then*

$$Prob(X \;\geq\; \frac{7}{8}m) \;\geq\; \frac{1}{m+1}$$

Proof: As in the proof of Theorem 1.6, we assume for simplicity (and without loss of generality) that each clause in φ consists of three literals corresponding to three different variables.

Recall that $E(X) = \sum_x x \cdot Prob[X = x]$. We can decompose this sum into two components. The first component G will represent the values of X that are greater or equal to $E(X)$. The second component R will represent the rest (i.e. the other possible x-s). Thus

$$E(X) \;=\; \sum_{x\in G} xProb[X = x] \;+\; \sum_{x\in R} xProb[X = x]$$

The possible values that X can get are integers. Let T be the largest integer that is smaller than $E(X)$. Using the definition of T and the fact that X is an integer, we have that $X < E(X)$ implies $X \leq T$. Thus:

$$E(X) \;\leq\; m \cdot Prob(X \geq E(X)) \;+\; T \cdot Prob(X < E(X)) \tag{1.3}$$

On the other hand

$$Prob(X \geq E(X)) \;+\; Prob(X < E(X)) \;=\; 1$$

Denote by α the quantity $Prob(X \geq E(X))$. Then $Prob(X < E(X)) = 1 - \alpha$. Hence, plugging the appropriate values into Equation 1.3 above, we obtain

$$E(X) \leq m\alpha \ + \ T \cdot (1 - \alpha) \tag{1.4}$$

Recalling the definition of T, we have

$$T \ = \ \lceil E(X) - 1 \rceil \ \leq \ E(X) - \frac{1}{8}$$

where the inequality holds because $E(X) = \frac{7}{8}m$, which equals $a + \frac{b}{8}$ for some non-negative integers a and b so that $b \leq 7$. (Extra help: if $b = 0$ then $\lceil a + \frac{b}{8} - 1 \rceil = a - 1 < (a + \frac{b}{8}) - \frac{1}{8}$, whereas if $b > 0$ then $\lceil a + \frac{b}{8} - 1 \rceil = a \leq (a + \frac{b}{8}) - \frac{1}{8}$.) Thus, Equation 1.4 yields:

$$E(X) \leq m\alpha \ + \ (E(X) - \frac{1}{8}) \cdot (1 - \alpha) \tag{1.5}$$

By a sequence of manipulations, we get the following sequence of equivalent conditions:

$$\alpha E(X) \ \leq \ m\alpha \ - \ \frac{1 - \alpha}{8} \tag{1.6}$$

$$\frac{1}{8} \ \leq \ (m - E(X) + \frac{1}{8}) \cdot \alpha \tag{1.7}$$

$$\alpha \ \geq \ \frac{1}{8m - 8E(X) + 1} \ = \ \frac{1}{8m - 7m + 1} \tag{1.8}$$

We conclude that $\alpha \ \geq \ \frac{1}{m+1}$, and the claim follows. ■

Having proved the above claim, we can now present a randomized algorithm which will get a formula φ as its input and output an assignment to the variables of φ that will satisfy $\frac{7}{8}$ of its clauses with exponentially high probability. The algorithm is as hinted above: it will selected at random truth assignments to the variables of φ until finding one as desired (which happens with probability at least $1/|\varphi|$ at each attempt). In fact, we prefer to present a version in which the number of attempts is bounded although the algorithm fails with small probability. The algorithm is as follows:

Repeat Steps 1–3 for $|\varphi|^2$ times

1. **select** a random assignment for the formula φ
2. **check** how many clauses of φ the chosen assignment satisfies
3. **if** the assignment satisfies at least $\frac{7}{8}$ of the clauses **then** output the assignment

Each iteration of the algorithm takes polynomial time, and so does the entire algorithm. Using the Claim 1.3.1 the probability that the algorithm will succeed in each single iteration is at least $\frac{1}{m+1}$, and this is at least $\frac{1}{|\varphi|}$ (since $|\varphi| > m$). Thus, the probability that the algorithm will fail in one iteration is less or equal to $1 - \frac{1}{|\varphi|}$. It follows that the probability that we fail in all $|\varphi|^2$ iterations is less or equal to $(1 - \frac{1}{|\varphi|})^{|\varphi|^2}$ which is approximately $e^{-|\varphi|}$. We conclude that the probability that algorithm succeeds is at least $1 - e^{-|\varphi|}$.

Bibliographic Notes

The Probabilistic Method is the topic of an excellent book by Alon and Spencer [10], and the reader is strongly encourage to learn more about it from that source.

Lecture 2

Laws of large numbers and Existence of Expanders

Notes taken by Shai Mor and Erez Waisbard

Summary: The focus of the lecture is various laws of large numbers. These laws bound the probability that a random variable deviates far from its expectation. Motivated by the problem of approximating the average of a function, we present stronger bounds than we had in the previous lecture. Moving from pairwise independent sampling to $2k$-wise independent sampling and then to totally independent sampling makes the the error probability exponentially vanishing (instead of linearly vanishing). We also define families of graphs called expanders, and prove the existence of a family of 3-regular expanders (using the probabilistic method).

2.1 Introduction

2.1.1 A motivating application

Loosely speaking the law of large numbers tells us that we can give a good estimation for the average of a bounded function by sampling enough points. Lets say we have a bounded function $f : \{0,1\}^n \to \{0,1\}$. It is infeasible for us to compute

$$\bar{f} \stackrel{\mathrm{def}}{=} \frac{1}{2^n} \sum_{x \in \{0,1\}^n} f(x)$$

in time which is polynomial in n, however if we sample enough points we can give a good estimation that will be correct with high probability.

With respect to such sampling, we can associate a random variable X_i withe the value of f obtained at each sample point. Clearly each X_i has expected value $\bar{f}$, and if we take enough samples then the average of the corresponding X_i's is very likely to be close to $\bar{f}$. Our focus here is on quantitative statements of the above intuition.

In the previous lecture we gave the following bound for a set of pairwise independent random variables $\{X_1 \ldots X_m\}$:

$$Prob\left[\left|\left(\frac{1}{m}\sum_{i=1}^{m} X_i\right) - \left(\frac{1}{m}\sum_{i=1}^{m} E(X_i)\right)\right| \geq \delta\right] \leq \frac{max_i\{Var(X_i)\}}{m\delta^2} \tag{2.1}$$

We see that the more X_i's we have, the better our estimation gets. This is really what the law of large numbers is all about. It says that if you give me more random variables of the same distribution then I can give a better estimation for the average of their expectation. A crucial point is that they have to be at least pairwise independent. This property enables us to get a good error bound. We make two observations looking at the above inequality:

1. There is a tradeoff between the deviation we want to bound and the error probability. Naturally we would like to be able to improve the tradeoff by both reducing the deviation and dropping the error probability.
2. The above bound gives us an error probability that is linearly dropping in the number of samples (i.e., m).

By imposing a stronger than pairwise independence restriction on the random variables, we can get a stronger dependence of the error-bound on the number of samples. Loosely speaking, the error probability is exponentially vanishing with the level of independence (between the random variables).

2.1.2 Independence of random variables

We now define independence between random variables. In the previous lecture we defined pairwise independence. We now define total independence between random variables.

Definition 2.1 *Let $X_1 \dots X_m$ be random variables. We say that they are* totally independent *if for every m values $v_1 \dots v_m$, the following holds*

$$Prob[X_1 = v_1 \wedge \dots \wedge X_m = v_m] = \prod_{i=1}^{m} Prob[X_j = v_j]$$

Total independence and pairwise independence are two types of notions of independence of random variables. The notion of independence can be extended to express independence of subsets of certain size of random variables.

Definition 2.2 *Let $X_1 \dots X_m$ be random variables. We say that they are* t-wise independent *if for every subset of t of them, $X_{i_1} \dots X_{i_t}$, and for every values $v_1 \dots v_t$, the following holds*

$$Prob[X_{i_1} = v_1 \wedge \dots \wedge X_{i_t} = v_t] = \prod_{i=1}^{t} Prob[X_{i_j} = v_j]$$

In other words, $X_1 \dots X_m$ are t-wise independent, if every t of them are totally independent. Another (equivalent) way to define independence between random variables is by saying that the probability for a random variable to have some value is not changed even if we know the values of the other random variables.

Definition 2.3 *Let $X_1 \dots X_m$ be random variables. We say that they are* t-wise independent *if for every subset of t of them, $X_{i_1} \dots X_{i_t}$, and for every values $v_1 \dots v_t$, the following holds*

$$Prob[X_{i_1} = v_1 | X_{i_2} = v_2 \wedge \dots \wedge X_{i_t} = v_t] = Prob[X_{i_1} = v_1]$$

We now see that pairwise independence is simply $t-$wise independence for $t = 2$ and total independence is $t-$wise independence for $t = m$. Clearly it holds that $(i + 1)$-independent random variables are also i-independent random variables. The question one can ask at this point is whether it is really a stronger notion? The following example shows that it is.

Example 2.4 *Let X and Y be two independent random variables and let $Z = X \oplus Y$ be a third random variable. Clearly every two of these random variables are independent, however the three of them are dependent - knowing the value of any two of them immediately implies value of the third.*

2.2 Stronger laws of large numbers

In the previous lecture we initiated the study of techniques for bounding the probability that a random variable deviates far from its expectation. In this lecture we focus on a technique for obtaining considerably sharper bound on such probability. We now come to define the moments of a random variable as a measurement for its concentration. Taking advantage on the fact that the deviation of totaly independent random variables grows much slower than the deviation of dependent random variables we present the Chernoff bound who's error probability is exponentially vanishing. In addition, we present probability bound for the sum of k-wise independent random variables, where this bound vanish exponentially with k.

2.2.1 Moments

Loosely speaking the moments of a random are a measurement of its concentration. One type of moment is the variance, which was already defined in the last lecture. Other (higher) moments gives us a better measure for its concentration. We will be interested mostly with the moments of a normalized random variable.

Definition 2.5 *The t-centralized moment of a random variable X is*

$$M_t(X) = E((X - E(X))^t)$$

We can immediately see that $Var(X) = M_2(X)$.
We now come to use the bound on the moments to give a better error probability on the deviation of random variables.

2.2.2 Law for $2k$-independent trials

Theorem 2.6 *For $2k$-wise independent random variables $X_1 \ldots X_m$*

$$Prob\left[\left|\left(\frac{1}{m}\sum_{i=1}^{m} X_i\right) - \left(\frac{1}{m}\sum_{i=1}^{m} E(X_i)\right)\right| \geq \delta\right] \leq \frac{\max_i \left(\max_{t\geq 2}(M_t(X_i))^{2k/t}\right)}{\left(\frac{m\delta^2}{k^2}\right)^k}$$

Note that for the case that $k = 1$ this is exactly what we had in Eq. (2.1).
Proof: In order to prove the above theorem we first normalize the random variables. That is, we define

$$\bar{X}_i = X_i - E(X_i)$$

This technicality will play a crucial part in the proof because the expectation of a normalized random variable is 0.

We rewrite the probability in the claim in terms of the normalized random variable, using the fact that we raise it to an even power to get rid of the absolute value:

$$Prob\left[\left|\left(\frac{1}{m}\sum_{i=1}^{m}\bar{X}_i\right)\right| \geq \delta\right] = Prob\left[\left(\frac{1}{m}\sum_{i=1}^{m}\bar{X}_i\right)^{2k} \geq \delta^{2k}\right]$$

Applying the Markov inequality we get

$$\begin{aligned} Prob\left[\left(\frac{1}{m}\sum_{i=1}^{m}\bar{X}_i\right)^{2k} \geq \delta^{2k}\right] &\leq \frac{E\left((\frac{1}{m}\sum_{i=1}^{m}\bar{X}_i)^{2k}\right)}{\delta^{2k}} \\ &= \frac{E((\sum_{i=1}^{m}\bar{X}_i)^{2k})}{m^{2k}\delta^{2k}} \end{aligned}$$

We now focus on the numerator. Using the linearity of expectations we get

$$E\left((\sum_{i=1}^{m}\bar{X}_i)^{2k}\right) = \sum_{(i_1 \ldots i_{2k})\in[m]} E(\bar{X}_{i_1}\cdots\bar{X}_{i_{2k}})$$

where $[m] = \{1,\ldots,m\}$.
We now split the right sum to the sum over two disjoint subsets A and B, where A is a set of $2k-$long sequences in which some integer appears uniquely, and B is a set of $2k-$long sequences in which each integer appears with multiplicity at least 2. That is $(i_1 \ldots i_{2k}) \in A$ if there exist i such that a unique j satisfies $i_j = i$ and $(i_1 \ldots i_{2k}) \in B$ if for every j there exist $j' \neq j$ such that $i_j = i_{j'}$.

(Note that if we think of m as being much greater than k, we can expect most of the terms to be from the subset A. As we will see next, these terms will have no contribution to the total sum.)

$$\begin{aligned} \sum_{(i_1 \ldots i_{2k})\in[m]} E(\bar{X}_{i_1}\cdots\bar{X}_{i_{2k}}) &= \sum_{(i_1 \ldots i_{2k})\in A} E(\bar{X}_{i_1}\cdots\bar{X}_{i_{2k}}) + \sum_{(i_1 \ldots i_{2k})\in B} E(\bar{X}_{i_1}\cdots\bar{X}_{i_{2k}}) \\ &= 0 + \sum_{(i_1 \ldots i_{2k})\in B} E(\bar{X}_{i_1}\cdots\bar{X}_{i_{2k}}) \end{aligned}$$

The reason that the sum over A equals zero follows from the independence of the $\bar{X}_i$'s and using $E(\bar{X}_{i_j}) = 0$ for i_j which appears with multiplicity 1. That is, for $(i_1 \ldots i_{2k}) \in A$ so that $i_j \neq i_{j'}$ for every $j \neq j'$, we have

$$E(\bar{X}_{i_j}\prod_{j\neq j'}\bar{X}_{i_{j'}}) = E(\bar{X}_{i_j})E(\prod_{j\neq j'}\bar{X}_{i_{j'}}) = 0 \cdot E(\prod_{j\neq j'}\bar{X}_{i_{j'}}) = 0$$

where the first equality uses the fact that $\bar{X}_{i_j}$ is independent of $\prod_{j\neq j'}\bar{X}_{i_{j'}}$ (since all $i_{j'}$ are different from i_j).
We now need to bound the sum over B. We do it by counting the number of sequences of length $2k$ over $[m]$ in which each occurrence is with multiplicity at least 2 (which clearly upper bounds $|B|$). Let j denote the number of integers occurring in the sequence. Since each such integer appears at least twice, j can not be larger than k, thus an upper bound on the number of such sequences is:

$$\sum_{j=1}^{k}\binom{m}{j}j^{2k} \ll m^k k^{2k}$$

Each of these sequences contributes

$$E(\bar{X}_{i_1}^{e_1}\cdots\bar{X}_{i_j}^{e_j}) = E(\bar{X}_{i_1}^{e_1})\cdots E(\bar{X}_{i_j}^{e_j}) = M_{e_1}(\bar{X}_{i_1})\cdots M_{e_j}(\bar{X}_{i_j})$$

where the e_j's are the number of occurrences of each i_j (and so $\sum_j e_j = 2k$). So we get

$$E((\sum_{i=1}^{m}\bar{X}_i)^{2k}) \;\leq\; m^k k^{2k}\cdot\max_i(\max_{e\geq 2}(M_e(\bar{X}_i)^{\frac{2k}{e}})$$

which in turn implies

$$\frac{E((\sum_{i=1}^{m}\bar{X}_i)^{2k})}{m^{2k}\delta^{2k}} \;\leq\; \frac{\max_i(\max_{e\geq 2}(M_e(X_i))^{\frac{2k}{e}})}{\left(\frac{m\delta^2}{k^2}\right)^k}$$

and the theorem follows. ■

This is clearly a much stronger bound than what we had before. Intuitively, if we consider k as a constant and consider X_i as a Bernoulli random variable, we get a very similar bound to the one we had before only now we get the k exponent in the denominator.

2.2.3 The Chernoff/Hoefding bound

Up until now the only restriction on the random variables was for them to be fully or partially independent. We now restrict ourselves to bounded random variables and come up with a tighter bound. The following theorem, which is called the Chernoff bound or the Hoefding bound, is presented here without proof.

Theorem 2.7 *For totaly independent random variables $X_1 \ldots X_m$, where $X_i \in [a,b]$*

$$Prob\left[\left|\left(\frac{1}{m}\sum_{i=1}^{m}X_i\right) - \left(\frac{1}{m}\sum_{i=1}^{m}E(X_i)\right)\right| \geq \delta\right] \;\leq\; 2e^{-\frac{2\delta^2}{(b-a)^2}\cdot m}$$

In the special case where $X_i \in [0,1]$, we get

$$Prob\left[\left|\left(\frac{1}{m}\sum_{i=1}^{m}X_i\right) - \left(\frac{1}{m}\sum_{i=1}^{m}E(X_i)\right)\right| \geq \delta\right] \;\leq\; 2e^{-2\delta^2\cdot m} \tag{2.2}$$

Clearly, Theorem 2.7 follows from Eq. (2.2); e.g., for X_i's as in Theorem 2.7, define $Y_i = (X_i - a)/(b-a)$ and apply Eq. (2.2).

Oded's Note: *Following are some remarks regarding the proof of Eq. (2.2). Firstly, we argue that it suffices to consider 0-1 random variables (intuitively, because they have the largest variance among random variables with range $[0,1]$). For 0-1 X_i's, denote $p_i = \Pr[X_i = 1]$ and $\bar{X}_i = X_i - p_i$. Without loss of generality we bound the probability that $\frac{1}{m}\sum_{i=1}^{m}\bar{X}_i$ is greater from δ. For any $\lambda > 0$, we have*

$$\begin{aligned}\Pr\left[\frac{1}{m}\sum_{i=1}^{m}\bar{X}_i \geq \delta\right] &= \Pr\left[e^{\lambda\sum_{i=1}^{m}\bar{X}_i} \geq e^{\lambda\delta m}\right] \\ &< \frac{E(e^{\lambda\sum_{i=1}^{m}\bar{X}_i})}{e^{\lambda\delta m}}\end{aligned}$$

Using the total independence of the X_i's we have $E(e^{\lambda \sum_{i=1}^m \bar{X}_i}) = \prod_{i=1}^m E(e^{\lambda \bar{X}_i})$. On the other hand, $E(e^{\lambda \bar{X}_i}) = p_i e^{\lambda \cdot (1-p_i)} + (1-p_i) e^{\lambda \cdot (0-p_i)} \leq e^{\lambda^2/8}$, where the last inequality holds for any $p_i \in [0,1]$ (and has elementary proofs unrelated to probability theory). Combining all the above, we get

$$\begin{aligned} \Pr\left[\frac{1}{m}\sum_{i=1}^m \bar{X}_i \geq \delta\right] &< \frac{(e^{\lambda^2/8})^m}{e^{\lambda\delta m}} \\ &= e^{\left(\frac{\lambda^2}{8} - \lambda\delta\right)\cdot m} \end{aligned}$$

Setting $\lambda = 4\delta$, we get $\Pr\left[\frac{1}{m}\sum_{i=1}^m \bar{X}_i \geq \delta\right] < e^{2\delta^2 \cdot m}$, and the claim follows.

2.2.4 Multiplicative Chernoff Bounds

Oded's Note: *Unfortunately, I did not manage to say the following in class.*

In some cases it is more useful to use "Multiplicative Chernoff Bounds" of the following form. For independent and indentically distributed X_i's in $[0,1]$ with $p = \Pr[X_i = 1]$, and for every $\gamma > 0$, it holds that

$$\Pr\left[\frac{1}{m}\sum_{i=1}^m X_i >, (1+\gamma)\cdot p\right] < e^{\gamma^2 p \cdot m/3} \quad (2.3)$$

$$\Pr\left[\frac{1}{m}\sum_{i=1}^m X_i <, (1-\gamma)\cdot p\right] < e^{\gamma^2 p \cdot m/2} \quad (2.4)$$

Note that by applying the standard (i.e., additive) Chernoff bound we get a probability bound of $e^{2(\gamma p)^2 \cdot m}$ in both cases. The Multiplicative forms are thus preferable for small values of p (e.g., for $p < 0.1$).

2.3 Application: Estimating the mean of a function

In this section we discuss probabilistic polynomial time computations that will do approximations to problems that require by their inherent nature exponential-time deterministic computations. Note that we regard computations as feasible only if their time of execution is at most of polynomial order of magnitude and infeasible if their time is of exponential order.

These approximations will rely heavily on the "Laws of Large Numbers" presented before. We will use *samples* from the problem's (i.e., function's) domain.

Given a boolean function $f : \{0,1\}^n \to [0,1]$, we denote f's mean value by $\bar{f}$. That is

$$\bar{f} \stackrel{\text{def}}{=} \frac{1}{2^n} \sum_{x \in \{0,1\}^n} f(x).$$

In order to exactly compute $\bar{f}$ one is required to take 2^n evaluations of f (i.e. sample f's domain in its entirety). Thus, a feasible estimation of $\bar{f}$ can not precisely follow its strict definition because the domain of function f is of exponential size in n. Fortunately there are "shortcuts" but inevitably these come with a "price tag". Here we shall introduce the notion of estimating (or approximating) $\bar{f}$ in (probabilistic) polynomial time.

The problem we face is how to give a satisfactory probabilistic approximation of $\bar{f}$ by means of a polynomial computation.

Definition 2.8 *An* (ϵ, δ)-sampler*, S, is a probabilistic polynomial-time oracle machine that when given oracle access to the function f satisfies*

$$Prob[|S^f(1^n, \epsilon, \delta) - \bar{f}| > \delta] < \epsilon.$$

In the above, $S^f(1^n, \epsilon, \delta)$ denotes the output distribution of the oracle machine S on input 1^n, ϵ and δ, and oracle access to the function f. The above definition means that the probability that the output of the sampler deviates from the actual value, $\bar{f}$, by more than δ is no more than ϵ.

The **question** we now face is "How many oracle access are needed in order to give a (ϵ, δ) approximation to $\bar{f}$"?. That is, what is the query complexity of (ϵ, δ) samplers.

Theorem 2.9 (a constructive upper bound) *The query complexity of (ϵ, δ) samplers is at most $O(\frac{1}{\delta^2} log(\frac{1}{\epsilon}))$.*

Proof: In order to assert the above bound on the number of queries we suggest the following

- Take $m \stackrel{\text{def}}{=} O(\frac{1}{\delta^2} log(\frac{1}{\epsilon}))$ independent uniform samples on f's domain, denoted by $s_1, s_2, ..., s_m \in \{0,1\}^n$.
- Output $\frac{1}{m}\sum_{i=1}^m f(s_i)$ which will be implement as follows:

  For i=1 to m do: Make a oracle call to f with input s_i, obtain $v_i = f(s_i)$, compute $\frac{1}{m}\sum_{i=1}^m v_i$ and output it.

We will prove that the estimation we gave on the number of samples is indeed tight from bellow (i.e., m samples are necessary for an (ϵ, δ) approximation).

> Oded's Note: *The probability space consists of m-long sequences of n-bit strings; that is, of all possible choices of $\overline{s} = (s_1, ..., s_m) \in (\{0,1\}^n)^m$. We define m random variables $X_1, X_2, ..., X_m$ so that $X_i(\overline{s}) = f(s_i)$. Thus, X_i actually depends only on s_i, and so these random varibales are clearly independent.*

Let $X_1, X_2, ..., X_m$ be m random variables, such that $X_i = f(s_i)$. These random variables are clearly independent of one another, and each ranges in $[0,1]$ and has expectation that equals $\bar{f}$. Applying Chernoff's Bound, we get

$$Prob\left[\left|S^f(1^n, \epsilon, \delta) - \bar{f}\right| > \delta\right] = Prob\left[\left|\frac{1}{m}\sum_{i=1}^m X_i - \bar{f}\right| > \delta\right] < 2e^{-2\delta^2 \cdot m}$$

Thus, the bound is exponentially vanishing in $\delta^2 m$. We compare this bound to ϵ and derive an bound on the number of samples, m, which are required by our (ϵ, δ) approximation algorithm; that is, $2\exp[-2\delta^2 \cdot m] = \epsilon$ which implies

$$m = \frac{1}{2\delta^2} \cdot \log(\frac{2}{\epsilon}).$$

as required. ■

Notice that the bound on the number of sufficient samples does NOT dependent on n, the function f's domain size. This means that the bound does not distinguish between functions of large domain and functions of relatively smaller domain.

The upper bound (on the number of samples that are sufficient for an (ϵ, δ) approximation) given in Theorem 2 is tight up to a multiplicative factor. That is:

Theorem 2.10 (a lower bound) *The query complexity of (ϵ, δ) samplers is at least $\Omega(\frac{1}{\delta^2} log(\frac{1}{\epsilon}))$.*

2.4 Existence of Expanders via the Probabilistic Method

In this section we will show the existence of expanders via the "The Probabilistic Method", which was introduced in previous lecture. This is a combinatorial tool for demonstrating the existence of combinatorial objects. The example we will focus on is a *probabilistic proof of existence* for an expander (i.e., a family of graphs with certain properties). By a probabilistic proof we mean that we do not know how to explicitly find the expander we are looking for and the proof of its existence is implicit. We merely argue that a random object that we construct is not an expander is strictly smaller than 1. This is enough to obtain the theorem.

2.4.1 Expander Graphs

In order to start speaking about expanders, we will need some definitions from graph-theory to assist us in what follows.

Definition 2.11 *In a graph $G(V, E)$ a* neighborhood, *Γ, of a vertex $v \in V$ is the set of all adjacent vertices to v (vertices that are connected to v by an edge). A neighborhood is described by the following notation :*

$$\Gamma(v) \stackrel{\text{def}}{=} \{u : (v, u) \in E\}.$$

Note also that one can also extend the above definition for an arbitrary subset $V' \subseteq V$ (referring to the neighborhoods of a subset of vertices) by

$$\Gamma(V') \stackrel{\text{def}}{=} \{v \in V \mid \exists (v', v) \in E \; for \; some \; v' \in V'\}.$$

Thus,

$$\Gamma(V') = \bigcup_{v' \in V'} \Gamma(v').$$

Continuing with the spirit of the above definition we shall now proceed with defining the outer section of a vertices set

Definition 2.12 *For a set S,* the boundary *of the set, $\Gamma'(S)$, is the set of vertices that are adjacent to some vertex in S and are not in S themselves. That is:*

$$\Gamma'(S) \stackrel{\text{def}}{=} \Gamma(S) \setminus S.$$

> Oded's Note: *The following two motivational paragraphs are due to the scribes themselves. I did not motivate the definition of expander graph beyond saying that they will play an important role in some future lectures.*

A very import class of sparse graphs are expander graphs. Among other things, they can be considered as a model for a network with certain desirable properties. Basically an expander has the property that every subset of its vertices has a large set of neighbors. That implies that any pair of vertices are connected by a short path. Furthermore, removing random edges in the graph (simulating local connection failures) does not reduce the property by much, so a network which is an expander can be considered as a fault-tolerant. We focus on expander graphs that are regular.

Definition 2.13 *A graph is d-*regular *if and only if every vertex has degree d.*

Now let us proceed to the definition of expander

Definition 2.14 *A d-regular graph $G(V,E)$ is a (d,c)-*expander *or has a c-*expansion *(for some $c > 0$) if and only if for every subset $V' \subseteq V$ of size $|V'| \leq |V|/2$ it holds that*

$$|\Gamma'(V')| \geq c|V'|.$$

To see that an expander makes a good network, suppose that you want to route a message from a vertex a to a vertex b in G. Every vertex has d neighbors. By the properties of an expander, there are at least $(1+d)(1+c)$ vertices at distance no more than 2 from a. Going further away, there will be at least $(1+d)(1+c)^k$ vertices at distance no more than $k+1$ from a. We can continue expanding from a until the reachable set of vertices V_a has more than $|V|/2$ vertices. The vertex b may not be among them, but if we expand from b in the same way, we eventually obtain a set V_b of more than $|V|/2$ vertices reachable from b. The sets V_a and V_b both contain more than $|V|/2$ so they must overlap. The overlap contains vertices on the path from a to b. In this way, we have shown that for any pair of vertices a to b, there is a path of length no more than $2(k+1)$ from a to b, where $k = \log_{1+c}|V|/2$. As one can observe from the expression for k, the larger the value of c the shorter the expected path between any two vertices.

The notion of an expander will usually be employed in the context of taking c as a small constant and of V as a huge vertex set (because we are interested in cases where we face very large graphs). We will generalize the above definition to a collection of graphs, referred to as a "family", each of which have an expansion.

The above definition of a (d,c)-expander extends naturally a family, F, of graphs $F = \{G_N\}_{N\in S}$, where G_N is an N-vertex graph and S is an infinite subset of N. That is, for each N in S there exists a single graph in the family that has N vertices and all the graphs in the family are (d,c)-expanders.

2.4.2 On suitable Sets of graph sizes

Let us define a function $next_S$ so that for every integer m:

$$next_S(m) \stackrel{\text{def}}{=} min_{n\in S}\{n \geq m\}.$$

That is, $next_S$ returns the smallest integer that is at least as big as m. The growth rate of $next_S$ determines the suitability of S to be used as graph sizes in families of graphs. The criterion that we will adopt in order to examine the suitability of a given set to be used in this context is that the output of the $next_S$ function is bound by a polynomial on its input. Our reason for adopting such a criterion is because upon given an m, our purpose is to construct a graph, G, with $n \geq m$ vertices. Typically, we will need to embed m objects as values of G. The difficulty is that we don't want the graph, G, to be too big because our measure of computational complexity for any given task/algorithm on a graph is based on the size of that graph.

Some typical examples of sets S that are practical and useful in various applications are the following:

- $S = \{2n : n \in \mathsf{N}\}$. Here $next_S(m) \leq m+1$.
- $S = \{Primes\}$. Here $next_S(m)$ is quite complex but certainly at most $2m$ (because each interval $[m, 2m]$ must contain a prime).
- $S = \{n^2 : n \in \mathsf{N}\}$. Here

$$\begin{aligned} next_S(m) &= \lceil\sqrt{m}\rceil^2 \\ &< m + 2\sqrt{m} + 1 \\ &< m + o(m). \end{aligned}$$

Even for $S = \{2^n : n \in \mathsf{N}\}$, We have

$$\begin{aligned} next_S(m) &= 2^{\lceil \log_2(m) \rceil} \\ &\leq 2m. \end{aligned}$$

On the other hand, bad examples correspond to very-sparse sets such as $S = \{2^{2^{2^n}} : n \in \mathsf{N}\}$. Here, for some m,

$$\begin{aligned} next_S(m) &= 2^{2^{2^{\log\log\log(m)+1}}} \\ &= 2^{2^{2\log\log m}} \\ &= 2^{(\log m)^2} \\ &= m^{log(m)} \end{aligned}$$

and that is more than any polynomial in m.

2.4.3 On the existence of 3-regular expanders

Now, we proceed to providing a probabilistic proof of the existence of expanders. We will use the following graph-algorithmic definition :

Definition 2.15 *Given a graph $G(V, E)$ a* matching *of G is a subset of the edges of G such that every vertex appears in at most one edge of the subset. A* perfect matching *is a matching where all vertices in G appear.*

Now we get to the main theorem of our discussion about expanders.

Theorem 2.16 *There exists a constant $c > 0$ for which, for all sufficiently large even N, there exists an N-vertex, 3-regular graph with expansion c.*

Proof: We use the probability method. We will show that for sufficiently small $c > 0$ and all sufficiently large (even) N, with high probability, a random 3-regular N-vertex graph has expansion c.

We will generate a random 3-regular graph G by selecting three random perfect matchings, denoted G_1, G_2 and G_3. To generate a random undirected graph G_1 with $N = 2K$ vertices and K edges, we will select edges by *selecting uniformly at random* a pair of distinct vertices, and removing them from the pile of available vertices (so our selection of a pair is a probabilistic event which is dependable on previous selections!). We will continue this selection of pairs of vertices until we exhaust the entire set of N vertices. When we are done, we actually have a perfect matching on G (i.e., K edges where created, every vertex is covered once). We will independently repeat this process for three times, thus getting three graphs $\{G_1, G_2, G_3\}$ and three very-possibly disjoint sets of edges, matchings, (no edge will repeat itself unless the same pair was selected more than once, which is highly unlikely as N becomes very large). Then we combine our three graphs into one possibly multi-graph G which is 3-regular (every vertex has no more then three neighbors, if there are less than three then there exists a neighbor to which it has a multi edge (two or three parallel edges)). We want to prove that *the probability that graph G has expansion c is bigger than zero (and moreover it's close to 1).* We will prove this by showing that the probability of the complement event (that there is no such expansion c of G) is strictly less than 1 (and actually close to 0). That is, the probability that there exists a subset of vertices that has too small boundary and so does not permit the existence of an expansion c of G is strictly less than 1. We shall bound the probability of the union of all these "bad events" by means of the Union Bound argument:

Proposition 2.4.1 (Union Bound): *For every series of sets* $\{A_i\}$ *and every random variable* X*:*

$$Prob[X \in \bigcup_{i=1}^{n} A_i] \leq \sum_{i=1}^{n} Prob[X \in A_i].$$

We now get to the details of the actual proof. Let $N = 2K$ be the number of vertices of graph G_N. We will evaluate the probability of the complement event to the event that we are interested in; that is, the probability that G_N does not have expansion c. That is, we look at the probability of the following (complementary) event:

$$\text{there exists a subset } V' \text{ s.t. } |V'| \leq K \text{ and } |\Gamma'(V')| < c|V'|. \tag{2.5}$$

By the union bound, the probability that the event in Eq. (2.5) occurs is at most

$$\sum_{V':|V'|\leq K} Prob[|\Gamma'(V')| < c|V'|] \tag{2.6}$$

Again, by the union bound, Eq. (2.6) is bounded from above by

$$\sum_{V':|V'|\leq K} \; \sum_{V'':|V''|=c|V'|} Prob[\Gamma'(V') \subset V''] \tag{2.7}$$

We give an upper bound on Eq. (2.7) by considering all possible V' of size i and all V'' of size ci, where i goes from 1 to K (i.e., we are enumerating over all possible sizes of the two sets). Clearly, it suffices to consider in the second sum only the sets V'' that are disjoint of V'. Thus, we get the bound

$$\sum_{i=1}^{K} \binom{N}{i} \cdot \binom{N-i}{ci} \cdot Prob[\Gamma'(V'_0) \subset V''_0]$$

where V'_0 and V''_0 are arbitrary fixed disjoint sets of sizes i and ci, respectively. Equivalently, we may bound

$$\sum_{i=1}^{K} \binom{N}{i} \cdot \binom{N-i}{ci} \cdot Prob[\Gamma(V'_0) \subset V'_0 \cup V''_0]$$

which, using $V'_0 = [i] \stackrel{\text{def}}{=} \{1, ..., i\}$ and $V''_0 = \{i+1, ..., i+ci\}$, is upper bounded by

$$\sum_{i=1}^{K} \binom{N}{i} \cdot \binom{N}{ci} \cdot Prob\left[\Gamma([i]) \subseteq [i+ci]\right] \tag{2.8}$$

We now focus on bounding the probability that $\Gamma([i]) \subseteq [i+ci]$. That is, we bound the probability that in all three perfect matchings, the vertices in $[i]$ are matched to vertices in $[i+ci]$. Focusing on each of these three perfect matchings, the probability that in a single perfect matching all the vertices in $[i]$ are matched to vertices in $[i+ci]$ is at most

$$\frac{i+ci-1}{N-1} \cdot \frac{i+ci-3}{N-3} \cdots \frac{i+ci-i+1}{N-i+1} \;=\; \prod_{j=1}^{i/2} \frac{i+ci-2j+1}{N-2j+1}$$

where the bound is obtained by considering a process of iteratively matching $i/2$ vertices in $[i]$, where in each step we match the next unmmatched vertex and consider the probability that it is matched to one of the remaining unmatched vertices in $[i+ci]$. By algebraic manipulations we get:

$$\begin{aligned}
&\prod_{j=1}^{i/2} \frac{i+ci-2j+1}{N-2j+1} \\
&< \prod_{j=0}^{(i/2)-1} \frac{i+ci-2j}{N-2j} \\
&= \frac{2^{i/2} \cdot \prod_{j=0}^{(i/2)-1} \frac{i+ci}{2} - j}{2^{i/2} \cdot \prod_{j=0}^{(i/2)-1} \frac{N}{2} - j} \\
&= \frac{\binom{(i+ci)/2}{i/2}}{\binom{N/2}{i/2}}
\end{aligned}$$

Recall that the above line provides an upper bound on the probability that a single random perfect matching matches all vertices in $[i]$ to vertices in $[i+ci]$. Thus, the probability that is happens in all three perfect matchings used to construct the graph is the cube power of the above, and we get:

$$Prob\left[\Gamma([i]) \subseteq [i+ci]\right] < \left(\frac{\binom{(i+ci)/2}{i/2}}{\binom{N/2}{i/2}}\right)^3$$

Plugging this into Eq. (2.8), we obatin the bound

$$\sum_{i=1}^{K} \frac{\binom{N}{i} \cdot \binom{N}{ci} \cdot \binom{(i+ci)/2}{i/2}^3}{\binom{N/2}{i/2}^3} \tag{2.9}$$

Using the approximatiom $\binom{n}{\alpha n} = 2^{H_2(\alpha) \cdot n}$, where $H_2(\alpha) = \alpha \log_2(1/\alpha) + (1-\alpha)\log_2(1/(1-\alpha))$ is the binary entropy function, we get

$$\begin{aligned}
&\log_2\left(\frac{\binom{N}{i} \cdot \binom{N}{ci} \cdot \binom{(i+ci)/2}{i/2}^3}{\binom{N/2}{i/2}^3}\right) \\
&\approx H_2(i/N) \cdot N + H_2(ci/N) \cdot N + 3H_2(i/(i+ci)) \cdot \frac{i+ci}{2} - 3H_2(i/N) \cdot \frac{N}{2} \\
&= -(H_2(i/N) - 2H_2(ci/N)) \cdot \frac{N}{2} + 3H_2(1/(1+c)) \cdot \frac{i+ci}{2}
\end{aligned}$$

For a sufficiently small $c > 0$ (and $i \le N/2$), we have $H_2(i/N) < c' \cdot H_2(ci/N)$ for $c' \approx c\log(1/c)$, and $H_2((1/(1+c)) \approx H_2(c) \approx c\log_2(1/c)$. Plugging all this into Eq. (2.9), we obatin the bound

$$\sum_{i=1}^{K} 2^{-(1-c')H_2(i/2K) \cdot K + 2c' \cdot i}$$

which approaches zero when c does. The theorem follows. ∎

2.4.4 Non-existence of 2-regular expanders

Contrary to the previous theorem, the following theorem asserts exactly the opposite for 2-regular graphs

Theorem 2.17 *For all $c > 0$ and for all sufficiently large N, there exists NO $(2, c)$-expander with N vertices.*

Proof: Fix some arbitrary $c > 0$. Consider an N-vertex graph which is 2-regular. Without loss of generality consider a connected graph [if the graph is build out of components then we can choose all vertices in the smallest component (which contains at most $N/2$ vertices) and get that their boundary is null!]. Such a connected graph must be a closed cycle. Let us consider a connected subset of $N/2$ vertices. The boundary of such a subset is of size 2. In order to contradict the requirement for the existence of an expander, choose the number of vertices in the graph, N, such that $c \cdot N/2 > 2$ (i.e. $N > 4/c$ will do). ■

Bibliographic Notes

For more details on the proof of Chernoff Bound, the reader is referred to [10, Apdx. A]. The Multiplicative Chernoff Bounds are presented, proved and discussed in [34, Chap. 4]. Historical notes regarding the various forms of Chernoff Bound are also provided in [34, Chap. 4]. A proof of Theorem 2.10 (i.e., a lower bound on the sample complexity of (ϵ, δ)-samplers) can be found in [14].

Lecture 3

Small Pairwise-Independent Sample Spaces

Notes taken by Lev Faivishevsky and Sergey Khristo

We ...solemnly Publish and Declare that...
these are ...free and independent...
(Declaration of Independence)
Action of Second Continental Congress, July 4, 1776

Summary: Often in applications that use randomness (for example coin tosses) we do not need all these tosses to be independent, instead it may suffice that they are k-independent for some k. In the case of independent variables the probability space is at least of the size $|S|^m$, where m is the number of variables in the sequence and S is the sample space for an individual variable. In this lecture we present a construction for pairwise independent sample space of the size $\max(|S|, m)^2$ and its extension to k-wise independent case. This will allow us to *derandomize* the algorithm that finds assignment for 3CNF that satisfies at least 7/8 clauses, presented on the first lecture, just by means of exhaustive search.

3.1 Construction of k-wise independent sample space

Suppose we have sample space of the form $S^m = S \times S \times \ldots \times S$, where S is a finite set, with uniform distribution on it. Without loss of generality we can think of S as a set of integers $\{1, \ldots, |S|\}$.

Any element of S^m is a sequence of m elements from S, denoted $(s_1, \ldots, s_m)$. Clearly, every two components, taken alone, also have uniform distribution. Note, that if the pair $(x_1, x_2) \in X \times X$ is uniformly distributed on $X \times X$ then x_1 and x_2 are independent and uniformly distributed on X.

We are seeking for the subspace Ω that is substantially smaller than S^m, but the uniform distribution on Ω induces uniform distribution on any two coordinates. In the construction for Ω, presented below, we achieve $|\Omega| \simeq \max(|S|, m)^2$. This allow us, in some sense, to *save* randomness.

The construction that is described in detail below is a nice example but there are some technicalities that should be taken into account when working with it. They are discussed at the end of this lecture.

Suppose, for the beginning, that $m \leq |S|$. This restriction will be waived later. We will try to associate some algebraic structure to S; for example, by treating S as a finite field $\mathcal{F}$. Finite fields

exist only of sizes which are prime powers. The case when the size of S is not a prime power will be discussed later.
Let us now define such a sample space Ω:

$$\begin{aligned}\Omega &\stackrel{\text{def}}{=} \{\text{The set of all linear polynomials over } \mathcal{F}\} = \{ax+b|(a,b)\in\mathcal{F}^2\}\\ &\equiv \{(a\cdot 1+b, a\cdot 2+b, \ldots, a\cdot m+b)|(a,b)\in\mathcal{F}^2\}.\end{aligned}$$

The first line above gives a succinct representation of a set of m-long sequences over $\mathcal{F}$, whereas the second line gives an explicit representation of the same sequences.

Note that given a field $\mathcal{F}$, the space Ω can be easily sampled: you just pick two random elements from the field and get an element of Ω.

Let us denote by $\Pr_{x\in_R X}$ the probability over the set X with uniform measure on it. Write $p_{a,b}(i) = a\cdot i + b$ (the linear polynomial in i with coefficients a and b).

Claim 3.1.1 *For every $i \neq j$ from $[m] \subseteq \mathcal{F}$ and for every α and β from $\mathcal{F}$*

$$\Pr_{(a,b)\in_R\mathcal{F}^2}[p_{a,b}(i) = \alpha \wedge p_{a,b}(j) = \beta] = \frac{1}{|\mathcal{F}|^2}.$$

In other words, uniform measure on Ω results in pairwise independent uniform sample space.

Proof: Let us write formally

$$\Pr_{(a,b)\in_R\mathcal{F}^2}[p_{a,b}(i) = \alpha \wedge p_{a,b}(j) = \beta] = \frac{|\{(a,b)|\ p_{a,b}(i) = \alpha \wedge p_{a,b}(j) = \beta\}|}{|\mathcal{F}|^2}.$$

We have to show that the nominator equals one. Indeed, it is equal to the number of pairs (a,b) such that the following system of two equations is satisfied.

$$\begin{aligned}ai + b &= \alpha\\ aj + b &= \beta\end{aligned}$$

Thinking of a and b as variables and of i, j, α and β as constants, we conclude that if i is not equal j then this system has unique solution because the matrix

$$\begin{pmatrix} i & 1\\ j & 1\end{pmatrix}$$

has full rank. ■

Observe that the whole construction is independent of the field we use and one can use any field that he finds convenient. For example, if the size of S is prime then modular arithmetic is a good choice.

Note that the shown construction is not 3-wise independent. First, we do not have uniform (and independent) distribution on any *three* coordinates, since the there can be no solution for the system of the equations as in the proof of claim.

More generally, it is not possible to have a k-wise independent sample space that is smaller then $|S|^k$, because any fixed k coordinates are uniformly distributed and every value of $(x_1,\ldots,x_k)$ has some nonzero probability hence it is impossible to have less then $|S|^k$ elements in this set.

After a little meditating it becomes clear how to make our construction produce k-wise independent variables. For this it is enough to consider in the above definition of Ω polynomials of degree less than k. Specifically, we define

$$\begin{aligned}
\Omega' &\stackrel{\text{def}}{=} \{\text{The set of polynomials of degree less then } k \text{ over } \mathcal{F}\} \\
&= \{\overline{c} = (c_0, \ldots, c_{k-1}) : p_{\overline{c}} = \sum_{i=0}^{k-1} c_i x^i\} \\
&\equiv \{(p_{\overline{c}}(1), \ldots, p_{\overline{c}}(m)) : \overline{c} \in \mathcal{F}^k\}.
\end{aligned}$$

Again, the first lines give a succinct representation of a set of m-long sequences over $\mathcal{F}$, whereas the last line gives an explicit representation of the same sequences.

Recall that sample space is called k-wise independent if on any fixed k locations we have uniform distribution. Analogously to Claim 3.1.1, we have the following:

Claim 3.1.2 *For every different $i_1, \ldots, i_k$ from $[m] \subset \mathcal{F}$ and for every $\alpha_1, \ldots, \alpha_m$ from $\mathcal{F}$*

$$\Pr_{\overline{c} \in_R \mathcal{F}^k}[\forall j = 1 \ldots k, p_{\overline{c}}(i_j) = \alpha_{i_j}] = \left(\frac{1}{|\mathcal{F}|}\right)^k.$$

Proof: Similar to what we had in the previous claim,

$$\Pr_{\overline{c} \in_R \mathcal{F}^k}[\forall j = 1 \ldots k, p_{\overline{c}}(i_j) = \alpha_j] = \frac{|\{\overline{c} | \forall j = 1 \ldots k, p_{\overline{c}}(i_j) = \alpha_j\}|}{|\mathcal{F}|^k}$$

The nominator is equal the number of solutions to the following system:

$$\begin{cases}
c_0 + c_1 i_1 + \ldots + c_{k-1} i_1^{k-1} = \alpha_1 \\
\qquad \vdots \\
c_0 + c_1 i_k + \ldots + c_{k-1} i_k^{k-1} = \alpha_k
\end{cases}$$

Again one should think of c_i's as variables in the linear system.

To see that in fact there is exactly one solution let us write the system in the matrix form:

$$\begin{pmatrix}
1 & i_1 & i_1^2 & \ldots & i_1^{k-1} \\
1 & i_2 & i_2^2 & \ldots & i_2^{k-1} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & i_k & i_1^k & \ldots & i_k^{k-1}
\end{pmatrix}
\begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{k-1} \end{pmatrix}
=
\begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_{k-1} \end{pmatrix}$$

The matrix in the above system is a Van der Monde matrix and it is full rank (since $i_l \neq i_{l'}$ for every $l \neq l'$). ■

This construction is optimal in the size of Ω, but remember we are under the conditions that $m \leq |S|$ and that the size of S is must prime power (it allows us to treat S as a field and make all that claims about uniqueness of the solution of a linear system). The next section addresses these issues and shows how to deal with fields from the computational point of view.

3.2 Annoying technicalities

In this section we will cope with several technical pitfalls, that can potentially spoil our construction. These potential difficulties are:

1. The size of S may be not a prime power. It means that direct representation S as a finite field is impossible in this case.
2. What should we do if $m > |S|$?
3. How do we deal with fields, i.e. how can we define the sample subspace and how can we operate in it?

First of all, let's address the first problem. A general recipe is to consider substantially larger set S' with a "nice" embedding of S into S'. The meaning of nice is that each element of S appears in S' about the same number of times. So if we make a construction for S', it will remain valid for S also because there will be some good mapping from S' to S. More formally, we should have some mapping $\mu : S' \to S$ with the following properties:

$$\forall a, b \in S \quad |\mu^{-1}(a)| \approx |\mu^{-1}(b)|$$

The formula says that each image should have nearly the same number of preimages (it means that the mapping μ has to be "regular"). Actually, it is better that each element of S will have the same number of representatives in S'. If this condition doesn't hold then (exact) uniform distribution in S' wouldn't induce uniform distribution in S. Vice versa, if each element in S has exactly the same number of representatives in S' then uniform distribution in S' will induce uniform distribution in S via this mapping μ. It should be noticed that small discrepancy in this approximate equality like:

$$\forall a, b \in S \quad |\mu^{-1}(a)| = |\mu^{-1}(b)| \pm 1$$

doesn't disturb the validity of our constructions by much (especially if $|S'|/|S|$ is large; indeed, the bigger S' the better the approximation but the cost comes in terms of complexity which grows with S'). Hence whenever the initial space S doesn't satisfy us, we may switch to a larger S'. The same trick will help us further, in coping with the second problem.

Now we will give a similar solution to the second problem. In order to be more clear, we'll do it through an example. Suppose, we need to build sample space of 3-wise independent elements over S=$\{0,1\}$. More specifically, we want to construct the sample space of 3-wise independent sequences of length m with elements of S. We'll show that size of this sample has to be m^3 only. By Claim 3.1.2 (with $k = 3$) we would be able to do it if $|S|$ were bigger than m. Thus, we ought to take

$$S' = \{0, 1\}^l, \quad \text{where} \quad l = \lceil \log_2 m \rceil$$

Note, that this condition guarantees that $|S'| \geq m$, because

$$|S'| = 2^l = 2^{\lceil \log_2 m \rceil} \geq m$$

Finally one should mention, that $|S'|$ is a prime power. The size of the whole construction will be

$$|S'|^3 \approx (2l)^3 \approx m^3$$

The price that we have paid for the trick is that our constructed sample subspace has rather big size $\approx m^3$. The last step that should be done is to define the mapping $\mu : S' \to S$ that in our case implies

$$\mu : \{0,1\}^l \to \{0,1\}$$

The simple mapping may be just taking the first bit of an element in $\{0,1\}^l$ as the element's image in $\{0,1\}$. All needed properties of μ are satisfied. Generally, defining such a mapping is a relatively easy task, because we impose only the following restrictions on μ: it should be easy to compute and the mapping should be regular (remember, this means that each image should have nearly the same number of preimages).

The last technical problem to be treated is the handling of finite fields. We have assigned algebraic structure to constructed sample subspace. But how can we handle it? This issue clearly consists of several significant parts:

1. How do we build the sample space?
2. How do we explicitly write the elements of it?
3. How do we perform operations with the elements (like addition and multiplication)?

The answer to these questions amounts to providing an algorithm for constructing sample spaces. We will claim that this algorithm exists "modulo the field". In case of a field of prime cardinality, the meaning of the sentence "modulo the field" is that we are given the prime p. The general case will be considered later.

Claim 3.2.1 *Given prime p and number $m \geq p$ we may construct in* $\text{poly}(m)$*-time the sample space Ω:*

$$\Omega = \{\text{The set of all linear polynomials over field } \mathcal{F}\} \equiv \{ax+b|(a,b) \in \mathcal{F}^2\}$$

$$\equiv \{(a \cdot 1 + b, a \cdot 2 + b, \ldots, a \cdot m + b)|(a,b) \in \mathcal{F}^2\}.$$

Proof: In order to prove the claim we will explicitly give the algorithm that outputs p^2 such sequence of length m each. The algorithm will run in time polynomial in $p * m$.

The algorithm consists of two nested loops.

for a,b $\in F$
  for $i = 1$ to m
    output $(a \cdot i + b)$ mod $p)$.

We can see that running time of the algorithm is $O(p^2)$. Also if we want to generate a specific element and we know which one and a specific coordinate in it, we are also able to do it very efficiently. ■

Oded's Note: *What I tried to explain is that in general, we may be given an arbitrary set S and an integer m and need to construct a pairwise (or k-wise) independent space over S. In case $|S|$ is a prime and $|S| \geq m$, the above claim provides the answer, but in general $|S|$ may not be a prime (or even a prime power) or $m > |S|$. In such a case, we shall resort to the technique presented above; that is, construct a pairwise (or k-wise) independent space over some sufficiently large S', and map S' to S. But in such a case, we have to find S' (which has to be of prime (or prime power) cardinality) by ourselves. We may choose to find S' of prime cardinality or find S' of cardinality p^e where p is a* *small prime and e is an integer. In the first case, we face the problem of finding a large prime, and if the desired prime is huge then we are in real trouble (becuase we don't know of an efficient deterministic algorithm; see details below). In such a case, we turn to the second alternative, and use S' of cardinality p^e, when p is samll (e.g., $p = 2$). Such a prime poer is easy to find, but operating with a field of that cardinality is more involved.*

About the complexity of finding primes. Certainly, given an integer N, we can find a prime in the interval $[N, 2N]$ by using the trivial algorithm that scans all candiadtes and tries to factor each of them. This algorithm is deterministic and run in time polynomial in N, which is fine when N is not too big. A randomized algorithm can find such a prime in time polynomial in the length of the binary representation of N (i.e., $\text{poly} \log N$), but randomized algorithms are often not admissible in the context of constructing probability spaces.

We have shown that the sample space is explicitly constructible if we are given a prime p (i.e, as in a case we are given a set S and and integer $m < |S|$, and are asked to construct a sample space Ω for S^m, where $|S|$ is a prime). But what happens if we are not given p (i.e., as in case we are asked to construct such space when either $|S|$ is not a prime or $m > |S|$, and we proceed by constructing a sample space over a bigger S' of prime cardinality, which we have to find by ourselves)? In this case we face the problem of finding big primes (when the desired set is large). This is considered to be a difficult task (for deterministic algorithms). Typically, in such a case, it will be better to seek a field of cardinality p^e for some small prime p (e.g., 2) and big integer e. We'll consider this case in a while.

We should be able to represent explicitly elements of the field, that is trivial task in the case of a field of prime cardinality, and to make field operations, like addition and multiplication. Performing these operations is relatively easy when we are talking about a field of a prime cardinality (the operations are simply addition and multiplication modulo the prime).

Now we proceed to the case the cardinality of field is a prime power. We denote such a field as $GF(p^e)$, where p is a prime, and e is an integer bigger than 1. We again should give an answer on the following list of questions:

1. How do we represent elements of this field?
2. How do we operations in it, i.e. how can addition and multiplication be performed in such a field?

The set of elements of that kind of the field is typically represented by polynomials of degree smaller or equal to $e - 1$ over $GF(p)$. This is the canonical representation of this kind of the field. Here we assume that we already know how to operate with $GF(p)$. In particular, in the case of $GF(2)$ it's very easy. We are talking about numbers 0 and 1, and operations are addition and multiplication modulo 2.

Having defined the set of the elements, now we should equipped the field with operations. In this representation the field always comes with an irreducible polynomial of degree exactly e over $GF(p)$. A polynomial is called irreducible, if it has no non-trivial polynomial division. In other words, $Q(x)$ is irreducible over $F(p)$ if for every two polynomials $Q_1(x)$ and $Q_2(x)$ over $GF(p)$ if

$$Q = Q_1 \cdot Q_2$$

then one of these polynomials is actually a constant.

Now we are ready to define the operations. Addition is easily implemented as an addition of polynomials. Remember, that elements of our field are polynomials, and therefore we may add elements in the field directly as polynomials. We obtain the description of the field operation addition:

`addition`: *simple addition of polynomials*

We would like to introduce multiplication in the field also as a multiplication of the polynomials. Unfortunately, we cannot do it so easily, because product of two polynomials of degree less or equal to $e-1$ may be the polynomial of degree bigger than $e-1$, which is not a valid representation of a field element. The right way is to define this operation is as multiplication modulo irreducible polynomial, denoted Q. It means that we do multiplication in the following way: firstly, we calculate usual product of two polynomials and then we reduce the result modulo the irreducible polynomial $Q(x)$. To clarify the notion of reduction modulo polynomial, let's write explicitly $Q(x)$. Without loss of generality, we may assume that in $Q(x)$ coefficient of x^e is 1:

$$Q(x) = x^e + \sum_{i=0}^{e-1} q_i \cdot x^i \qquad (\text{in } GF(p))$$

Reduction modulo $Q(x)$ implies that the polynomial $Q(x)$ is considered to be a zero of the field $GF(p^e)$:

$$Q(x) = x^e + \sum_{i=0}^{e-1} q_i \cdot x^i = 0$$

Thus in $GF(p^e)$ we have

$$x^e = -\sum_{i=0}^{e-1} q_i \cdot x^i$$

Now whenever we encounter term x^e in result of product of polynomials, we may replace it by $-\sum_{i=0}^{e-1} q_i \cdot x^i$. Furthermore, we may replace x^{e+1} firstly by $x \cdot x^e$, and then substitute the above representation of x^e.

$$x^{e+1} = x \cdot x^e = x \cdot (-\sum_{i=0}^{e-1} q_i \cdot x^i) = -\sum_{i=0}^{e-1} q_i \cdot x^{i+1}$$

$$= -\sum_{i=0}^{e-2} q_i \cdot x^{i+1} - q_{e-1} \cdot x^e = -\sum_{i=0}^{e-2} q_i \cdot x^{i+1} + q_{e-1} \cdot \sum_{i=0}^{e-1} q_i \cdot x^i$$

So we have recursively represented x^{e+1} as a polynomial of degree less or equal to $e-1$. Obviously, we may represent any x^{e+k},(where $k = 1...e-2$) as a polynomial of degree less or equal to $e-1$ by the same recursive procedure.

Finally, we are able to represent any possible product of 2 polynomials degree less or equal to $e-1$ (it can be obviously a polynomial of degree up to $2e-2$) as a polynomial of degree less or equal to $e-1$. So now we complete our description of the field by introducing the operation of multiplication:

`multiplication`: *multiplication of polynomials modulo irreducible polynomial* $Q(x)$.

Unfortunately, there exists an number of difficulties. The problems are connected with the field representation of sample space. They don't appear when the set of field elements is somewhat small, because small primes can be found very easily. But if $|S|$ (or rather $|S'|$) has to be very big, then there are two alternative for field representation: either through a field of prime cardinality or through a field, whose size is a prime power. The former leads us to problem of finding big primes via a deterministic algorithm. Up to now it's not clear how we can solve it effectively. There exists probabilistic algorithms for big prime finding, but by definition it only gives a prime with *high probability*, while we would like to have a deterministic algorithm that produces a big prime.

The other option for the field representation; that is, as a field of prime power cardinality, seems to be less problematic. If p is small, then we just need to find an irreducible polynomial. There are solutions to this issue also, but they are beyond the scope of this lecture. We only give one special case for $GF(2^l)$. If l is of the form $l = 2 * 3^i$,where i is an integer, then the following polynomial is irreducible:

$$x^l + x^{l/2} + 1$$

Now, having resolved the technical difficulties of the construction, let's proceed to a nice application of this technique.

3.3 Application to 3CNF

Recall the algorithm for finding assignment for 3CNF formula that satisfies at least 7/8 clauses (presented on the first lecture). The algorithm works as follows: choose random assignment (uniformly) and test it. If it satisfies more than 7/8 of clauses then output the assignment, else choose another assignment. This algorithm works in expected polynomial time and requires a source of random bits. Here we present a modification that works in deterministic polynomial time.

Denote by m the number of variables in the input formula φ and by n the number of clauses. Let $\Omega \subset \{0,1\}^m$ be three-wise independent sample space. Using the technique described above, the size of Ω is of the order m^3. We claim that the expected number of clauses satisfied by a uniformly selected assignment in Ω equals $\frac{7}{8} \cdot n$ (exactly as is the case when selecting a totally random assignment (i.e., a uniformly selected assignment in $\{0,1\}^m$)):

Claim 3.3.1

$$\mathbf{E}_{\tau \in_R \Omega}(\text{number of clauses satisfied by } \tau) = \frac{7}{8} \cdot \text{number of clauses.}$$

Proof: Denote by I_i an indicator of the event that i'th clause is satisfied. Then,

$$\mathbf{E}_{\tau \in_R \Omega}[\text{number of clauses satisfied by } \tau] = \mathbf{E}(\sum_{i=1}^{n} I_i) = n \cdot \mathbf{E}(I_1),$$

where equality is due to linearity of expectation and to the fact that the I_j's are identically distributed. In order to compute $\mathbf{E}(I_1)$ recall that the distribution of individual letters in the assignment is three-wise independent and the fact that in each clause we have exactly three variables hence they are uniformly distributed over the set $\{0,1\}^3$. Thus,

$$\mathbf{E}(I_1) = \Pr[\text{clause is satisfied }] \cdot 1 + \Pr[\text{clause is not satisfied}] \cdot 0 = 7/8.$$

The claim follows. ■

Recall that $\Pr[X \geq E(X)] > 0$ for every random variable X. Letting X represent the number of clauses satisfied by an assignment uniformly selected in Ω, we conclude that Ω must contain an assignment that satisfies at least the expected number of satisfied clauses, (i.e., a 7/8th fraction of the clauses). Therefore, it is enough to scan the whole sample space Ω in search for such an assignment, and since the size of the space is only $O(m^3)$ we can do it efficiently.

3.4 Conclusion

The construction we have presented in this lecture is rather elegant. It enables dealing with k-wise independent elements in substantially small sample space. For the sake of simplicity, let's suppose that S is a field, and m is smaller than the size of S Thus we obtain k-wise independent sample space Ω and the size of the construction $|\Omega|$ is equal to $|S|^k$. This sample space can be explicitly constructed, and we have given an algorithm for it. One can write down all the elements of it in time:

$$O(m \cdot |S|^k) \cdot (time\ of\ field\ operations).$$

The size of our construction is (in some sense) optimal in terms $|S|$, because as we have discussed before, the size of construction of uniform over S variables with k-wise independency property cannot be less than $|S|^k$.

In case we are dealing with $S = \{0, 1\}$ and we follow the recipe which was described in Section 3.2, then the size of the sample space will be m^k. Let's consider the optimality of this size. More exactly, whether we can have the construction with the same property, but of substantially smaller size than m^k. The answer is no. There is a lower bound for k-wise independent sample space of sequences of length m over $S = \{0, 1\}$ that asserts that that the size of such a space must be at a least $m^{\lfloor k/2 \rfloor}$. That is, the basis is m and the exponent is k.

Bibliographic Notes

The constructions of pairwise and k-wise independent sample spaces (presented above) were re-discovered in computer science by Chor and Goldreich [16] and Alon, Babai and Itai [6], respectively. (They were originally discovered and appeared in the probability theory literature [29].)

Lecture 4

Small Pairwise-Independent Sample Spaces (cont.) and Hash Functions

Notes taken by Olga Grinchtein and Denis Simakov

Pluralitas non est ponenda sine neccesitate
(Entities should not be multiplied unnecessarily)
Ockham's Razor (William of Ockham)

Summary: A good construction of k-wise independent sample space should be small and convenient to use in applications. The construction presented in the previous lecture satisfies the first requirement (i.e., were small), but is rather complex, imposing a number of technical problems in application. In this lecture a second, much more convenient construction is presented, based on affine transformations. This construction yields small enough sample spaces, but only for pairwise independence ($k = 2$).

We also start discussing hash functions, show their relation to small pairwise independent sample spaces, and present a Hashing Lemma, connecting k-wise independency of hash functions to their "uniformness".

4.1 A second construction of small pairwise independent sample space

4.1.1 Motivation

Let's recall the problem we address. We have a finite set S and an integer m. We want to generate random strings of length m over S. So conceivably our sample space is S^m. But the size of this sample space is $|S|^m$ that is often much more than we can afford.

So naturally the task is to find set $\Omega \subset S^m$, which would be rather small and would possess the property of k-wise independency (k is one more integer parameter). This would enable us to use this small sample space Ω instead of huge S^m in problems, where k-wise independence suffices.

More specifically, *k-wise independence* means that if we watch strings through arbitrary k "windows", and these strings are uniformly chosen from Ω, then we will see in these windows uniform distribution, that is, for any distinct $i_1, \ldots, i_n$ and any $\alpha_1, \ldots, \alpha_k \in S$

$$\mathrm{Pr}_{e \in_R \Omega}\left[e_{i_1} = \alpha_1 \wedge \cdots \wedge e_{i_k} = \alpha_k\right] = \frac{1}{|S|^k}.$$

Concerning the size of Ω, typically we want it to be polynomial in $\max(|S|, m)^k$.

A first construction, solving this problem, we saw the last time. It is based on associating S with a finite field $\mathcal{F}$. Then k-wise independent Ω is constructed as the set of polynomials of degree less than k over the field $\mathcal{F}$. In explicit representation (in terms of elements of S^m) Ω is the set of values of such polynomials on some m distinct elements of the field:

$$\begin{aligned}\Omega &= \{\text{polynomials of degree} < k \text{ over field } \mathcal{F}\} \\ &= \left\{\Big(P(1), \ldots, P(m)\Big) \;:\; P \text{ – such polynomials}\right\}.\end{aligned}$$

This construction is quite satisfactory regarding independence (it works for any k) and size of Ω ($|\Omega| = |S|^k$, that is the smallest possible for k-wise independent space). But it is rather inconvenient for practical use due to a number of annoying technical difficulties that we have to overcome. We have to impose on S algebraic structure which can be complex (for example, finite fields), we have constraints on the size of S (it should be a prime power), and there is also inherent condition that $m \leq |S|$. Although we have demonstrated how to manage all these technicalities, we still would like more convenient methods.

In this lecture we will present *the second construction* of small pairwise independent space. It will have the following major differences, compared to *the first construction*:

+ less problematic;

− only for $k = 2$ (pairwise independence), and we don't know how to generalize it for arbitrary k.

So the second construction we present below is nice, and can be recommended for applications that require only pairwise independence. But whenever we need higher degree of independence, we still have our first construction at hand.

4.1.2 Construction

We'll assume that $S = \{0,1\}^n$ (if it's not the case, we can embed our sample space into set of binary strings). Unlike the first construction, we do not assume anything about the relation of m vs $|S|$.

Instead of polynomials over some field, in the second construction we will use affine transformations of boolean vectors. Affine transformation, given by pair (A, b), where A is $n \times l$ boolean matrix, b is n-dimensional boolean vector, maps all vectors $v \in \{0,1\}^l$ into vectors $Av + b \in \{0,1\}^n$ (all operations are done modulo 2).

So we define our new small sample space as follows:

$$\begin{aligned}\Omega &\stackrel{\text{def}}{=} \left\{\begin{array}{l}\text{set of all affine transformations} \\ \text{from } [m] = \{0,1\}^l \text{ to } S = \{0,1\}^n\end{array}\right\} \\ &\equiv \left\{(A, b) \;:\; \begin{array}{c}A \text{ is } n \times l \text{ boolean matrix,} \\ b \text{ is } n\text{-dimentional boolean vector}\end{array}\right\} \\ &= \left\{(Av_1 + b, \ldots, Av_m + b) \;:\; \begin{array}{c}\forall (A, b), \quad v_1, \ldots, v_m \text{ are fixed} \\ \text{distinct } l\text{-dimensional vectors}\end{array}\right\}.\end{aligned}$$

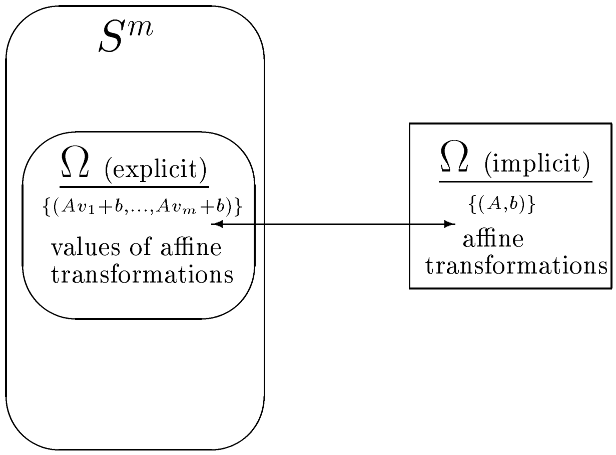
Remark. In this definition (and likewise in the 1st construction) Ω is given in two representations:

1. *Implicit representation* (as the set of affine transformations; or polynomials as before).

   This is how we define Ω, but in this representation Ω is not a subset of the initial sample space S^m. Implicit representation is a compact way of writing elements of Ω, and it allows us, for example, to generate all elements in Ω.

2. *Explicit representation* (as affine transformations, evaluated at m points; or evaluated polynomials). It is our implicit representation, now embedded into S^m. For Ω, written in this way, we can speak about pairwise independence.

Both representations are schematically presented in the picture:



Let's return to our new construction. Matrix A has n rows and l columns, vector b has length n. The sequence of m distinct vectors $\{v_i\}_{i=1}^m$ (each of length l) can be chosen, for example, to include binary numbers in canonical order (i.e. v_i is binary representation of number i).

So the sequence $(Av_1 + b, \ldots, Av_m + b)$ consists of m vectors of length n, and lies in $S^m = (\{0,1\}^n)^m$.

The parameter l is typically chosen to be $\lceil \lg_2 m \rceil$. We do not assume here any relation between m and n, so both cases $l \geq n$ and $l < n$ can happen.

Let's calculate the size of Ω. The number of binary $n \times l$ matrices A is 2^{nl}, the number of binary n-dimensional vectors b is 2^n, so

$$|\Omega| = \text{number of pairs } (A,b) = 2^{nl+n} \cong 2^{n(1+\lg m)},$$

$$|S^m| = (2^n)^m = 2^{nm} \gg |\Omega|.$$

This shows that we have achieved our goal of finding *small* sample space. Yet in the first construction we had size of k-wise independent sample space $|\Omega| = 2^{kn} = 2^{2n}$ for $k = 2$, which is considerably better. We'll address the issue of improving $|\Omega|$ in Sect. 4.2, but first let's check whether we indeed obtained a pairwise independent space.

4.1.3 Analysis: pairwise independence

Let's formulate the condition of pairwise independence as applied to the constructed Ω (from here on A is assumed to be $l \times n$ boolean matrix and b a boolean vector of length n):

$$\forall i \neq j \ \forall \alpha, \beta \in S \quad \Pr_{(A,b)}[Av_i + b = \alpha \wedge Av_j + b = \beta] = \frac{1}{|S|^2}.$$

It's convenient for us to formulate and proof another claim (clearly a more general one).

Claim 4.1.1 (pairwise independence of the construction)

$$\forall u \neq v \in \{0,1\}^l \quad \forall \alpha, \beta \in S = \{0,1\}^n$$
$$\Pr_{(A,b)}[Av + b = \alpha \wedge Au + b = \beta] = \frac{1}{|S|^2}.$$

Proof:

Step 1. We present a simplified expression for $\Pr_{(A,b)}[Av + b = \alpha \wedge Au + b = \beta]$.

Let's rewrite our probability in the following form:

$$\begin{aligned} &\Pr_{(A,b)}[Au + b = \alpha \wedge Av + b = \beta] \\ &\quad = \Pr_{(A,b)}[Av + b = \alpha] \cdot \Pr_{(A,b)}[Au + b = \beta | Av + b = \alpha] \end{aligned} \tag{4.1}$$

Consider the multipliers in the expression (4.1) one by one.

1. b is chosen uniformly, A and b are independent, whereas v and α are fixed, so $Av + b$ is uniformly distributed as well. So $\Pr_{(A,b)}[Av + b = \alpha] = \frac{1}{|S|}$. Let's show this more formally in two steps.

   First, for any fixed matrix A we have $\Pr_b[b = \alpha - Av] = \frac{1}{|S|}$ (because b is uniformly distributed on S).

   Second, A and b are independent, so from the fact that $\Pr_b[Av + b = \alpha] = \frac{1}{|S|}$ for any fixed A it follows, that this probability is also $\frac{1}{|S|}$ when A is random. This follows immediately by expanding as follows:

$$\begin{aligned} \Pr_{(A,b)}[Av + b = \alpha] &= \sum_{\mathcal{A}} \Pr_{(A,b)}\left[b = \alpha - Av \mid A = \mathcal{A}\right] \cdot \Pr_A[A = \mathcal{A}] \\ &= \sum_{\mathcal{A}} \Pr_b[b = \alpha - \mathcal{A}v] \cdot \Pr_A[A = \mathcal{A}] \\ &= \frac{1}{|S|} \sum_{\mathcal{A}} \Pr_A[A = \mathcal{A}] \\ &= \frac{1}{|S|} \end{aligned}$$

2. In $\Pr_{(A,b)}[Au + b = \beta | Av + b = \alpha]$ the event $Au + b = \beta$ is considered only under the condition $Av + b = \alpha$, so we can eliminate b by substitution $b = \alpha - Av$:

$$\begin{aligned} &\Pr_{(A,b)}[Au + b = \beta | Av + b = \alpha] \\ &\quad = \Pr_{(A,b)}[A(u - v) = \beta - \alpha | b = \alpha - Av] \\ &\quad = \Pr_{(A,b)}[Aw = \gamma | b = \alpha - Av] \end{aligned} \tag{4.2}$$

   where $w \stackrel{\text{def}}{=} u - v \neq 0$ and $\gamma \stackrel{\text{def}}{=} \beta - \alpha$ are fixed vectors.

   Now we see that event $b = \alpha - Av$ in (4.2) is totally irrelevant to $Aw = \gamma$. Hence $\Pr_{(A,b)}[Aw = \gamma | b = \alpha - Av] = \Pr_A[Aw = \gamma]$ (this can be shown by using Bayesian Rule[1]).

---

[1] $\Pr\left[Aw = \gamma \mid b = \alpha - Av\right] = \frac{\Pr[b=\alpha-Av \mid Aw=\gamma] \cdot \Pr[Aw=\gamma]}{\Pr[b=\alpha-Av]} = \Pr[Aw = \gamma]$,
because for every A $\Pr_b\left[b = \alpha - Av \mid Aw = \gamma\right] = \Pr_b[b = \alpha - Av] = 2^{-n}$

Thus, we obtained the following expression:

$$\Pr_{(A,b)}[Au + b = \alpha \wedge Av + b = \beta] = \frac{1}{|S|}\Pr_A[Aw = \gamma] \tag{4.3}$$

where w and γ are completely arbitrary, except for the fact that $w \neq 0$ (this is of course crucial). *Notice, that this result was obtained without using any assumptions about the internal structure of A (e.g., the distribution of its elements and so on).*

Now in order to prove our claim, we need only to show that $\Pr_A[Aw = \gamma] = \frac{1}{|S|}$.

Step 2. Prove that $\Pr_A[Aw = \gamma] = \frac{1}{|S|}$.

Since $w \neq 0$ there is a non-zero entry somewhere in w. Let the i'th element of w be not zero (i.e. be equal to 1).

Look at the i'th column of A. A is totally random, and its i'th column is also random and independent of all other columns of A. This column will add to Aw, consequently Aw is also fully random (uniformly distributed).

More formally we can write it in the following way. Let $A = (C_1, \ldots, C_l)$, where $C_1, \ldots C_l$ are columns of A. They are selected independently and uniformly of one another.

$$\Pr_A[Aw = \gamma] = \Pr_{C_1,\ldots,C_l}\Big[\sum_j C_j w_j = \gamma\Big],$$

$$\forall C_1, \ldots, C_{i-1}, C_{i+1}, \ldots, C_l \quad \Pr_{C_i}\Big[C_i = \gamma - \sum_{j \neq i} C_j w_j\Big] = \frac{1}{|S|}.$$

Now the same argument as in Step 1 works: the probability for every fixed tuple of values of non-i'th columns is $\frac{1}{|S|}$, hence it is not changed when $\{C_j\}_{j \neq i}$ are random.

Step 1 together with Step 2 complete the proof of the claim. We also noticed that conclusions of Step 1 were made without exploiting information about internal structure of matrix A, while Step 2 relies on the fact that the columns of A are independent.

■

So now we know that Ω is pairwise independent. Next we will consider improvement of the construction, that enables us to pass from $|\Omega| \sim 2^{nl}$ to $|\Omega| \sim 2^{n+l}$.

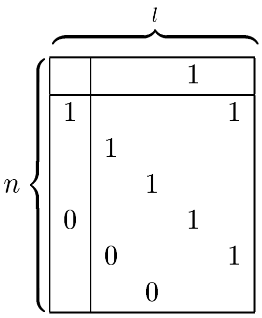
4.2 Improvement of initial construction (Toeplitz matrices)

4.2.1 Improved construction

To achieve smaller size of Ω we want to constrain the matrix space. For this purpose let's consider Toeplitz matrices, i.e. matrices with equal elements on all diagonals. The construction will be the same, but we will use a small subspace of matrices.

Definition 4.1 *$n \times l$ matrix $T = \{t_{i,j}\}$ is called Toeplitz if for every i, j $(1 \leq i \leq n-1, 1 \leq j \leq l-1)$ it holds that $t_{i,j} = t_{i+1,j+1}$.*

The following picture illustrates the notion of Toeplitz matrix.



A Toeplitz matrix is fully specified by $n + l - 1$ bits (for example, by values in the 1st row and the 1st column). So there are 2^{n+l-1} such matrices, which gives the size of our sample space $|\Omega| = 2^{l+2n+1}$ — comparable with 2^{2n} that we had in the first construction, and much less than 2^{nl+n} in the case of arbitrary affine transformations.

4.2.2 Analysis: pairwise independence

The proof of pairwise independence is very similar to the analogous proof for the case of arbitrary matrices (Claim 4.1.1). Part of it will be literally the same, and the rest will require some extra elaboration.

Here and below we will denote by T $n \times l$ boolean Toeplitz matrix and by b boolean vector of length n.

Claim 4.2.1 (pairwise independence of the Toeplitz matrices construction)

$$\forall u \neq v \in \{0,1\}^l \quad \forall \alpha, \beta \in S = \{0,1\}^n$$
$$\Pr_{(T,b)}[Tv + b = \alpha \wedge Tu + b = \beta] = \tfrac{1}{|S|^2}.$$

Proof:

Step 1. As we notices in the proof of Claim 4.1.1, Step 1 doesn't rely on the kind of matrices we deal with. Thus the results obtained there remain true if arbitrary matrix A is replaced by Toeplitz matrix T. So, we have "for free" the following conclusion (Eq. (4.3) of Claim 4.1.1):

$$\Pr_{(T,b)}[Tu + b = \alpha \wedge Tv + b = \beta] = \frac{1}{|S|}\Pr_T[Tw = \gamma]$$

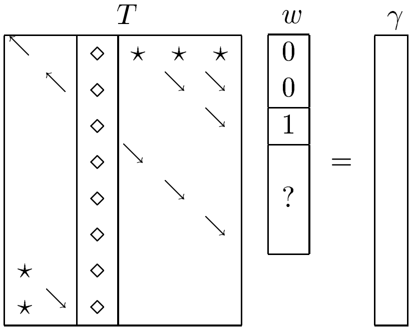
where w and γ are completely arbitrary, but $w \neq 0$.

Step 2. We show that $\Pr_T[Tw = \gamma] = \frac{1}{|S|}$, where $w \neq 0$ and γ are some fixed vectors (if $w = 0$ then $Tw = 0$ independently of random choice of T, so condition that $w \neq 0$ is crucial).

We cannot simply repeat the argument we had for the case of arbitrary random matrices A, because the columns of T are no longer independent.

Now instead of using an arbitrary non-zero element in w we select a particular one. Let it be the first one, let its index be i.

According to this given i (recall that w and γ are fixed, and only T is random) we choose specific representation of T: the last $i-1$ elements of the 1st column, all elements of the i'th column, and the last $l-i$ elements of the 1st row, as described in the picture (arrows describe how values "spread" over Toeplitz matrix):



So we can describe T by $\diamond$ and $\star$ parameters: $T \equiv (\diamond, \star)$, where $\diamond$ is the i'th column in T, $\star$ – the other part of the representation. These two parts of representation are independent from each other (for uniformly chosen T). We will show below a procedure, which for a fixed $\star$ determines $\diamond$ uniquely (from the equation $T(\diamond, \star)w = \gamma$). This implies that for any value of $\star$, the conditional probability $\Pr[Tw = \gamma | \star]$ equals $\frac{1}{|S|}$ (because $Tw = \gamma$ for given $\star$ determines a unique column $\diamond$ out of the $2^l = |S|$ possible ones in the residual probability spcae). This fact combined with the independence of $\star$ and $\diamond$ will give us what we need: $\Pr[Tw = \gamma] = \frac{1}{|S|}$ (by the same argument as in the Claim 4.1.1).

To complete the proof we only need to describe the procedure of solving equation $T(\diamond, \star)w = \gamma$ with respect to $\diamond$.

We will solve this equation row-by-row from top to bottom. First $i-1$ columns of T don't influence Tw (because they correspond to elements of w that are zero), so we can simply ignore them. Also taking into account that $w_i = 1$, we can write equation $Tw = \gamma$ in the following detailed form:

$$\begin{cases} t_{1,i} + t_{1,i+1}w_{i+1} + \ldots + t_{1,l}w_l = \gamma_1 \\ t_{2,i} + t_{2,i+1}w_{i+1} + \ldots + t_{2,l}w_l = \gamma_2 \\ \qquad \cdots \\ t_{j,i} + t_{j,i+1}w_{i+1} + \ldots + t_{j,l}w_l = \gamma_j \\ \qquad \cdots \\ t_{n,i} + t_{n,i+1}w_{i+1} + \ldots + t_{n,l}w_l = \gamma_n \end{cases} \tag{4.4}$$

The leftmost terms of these equations form precisely our unknown $\diamond = \{t_{1,i}, t_{2,i}, \ldots, t_{n,i}\}$.

Let's now look at the first row in (4.4). All values $t_{1,i+1}, \ldots, t_{1,l}$ are known (they are from $\star$, fixed in advance), hence we determine $t_{1,i}$ (the first element of $\diamond$) in the obvious way: $t_{1,i} = \gamma_1 - t_{1,i+1}w_{i+1} - \ldots - t_{1,l}w_l$.

But now, since we know the value of $t_{1,i}$, we can "propagate" it down the diagonal (i.e., it determines $t_{2,i+1}$). Then in the second row equation in (4.4) will remain only one unknow $t_{2,i}$.

In general, suppose we have found a unique specification for $t_{1,i}, \ldots, t_{j,i}$, $j < n$. Since we work with a Toeplitz matrix, this gives us values of all elements of T in the $(j+1)$'s row except for $t_{j+1,i}$ (black circles $\bullet$ in the picture below represent already known values of $\diamond$):

$$\begin{array}{rc|cccc|}
\cline{3-6}
1 \rightarrow & \bullet & \star & \star & \star & \star \\
 & \bullet & \searrow & \searrow & & \\
 & \bullet & \searrow & \searrow & \searrow & \\
j+1 \rightarrow & \diamond & \searrow & \searrow & \searrow & \searrow \\
 & \diamond & & & & \\
 & \diamond & & & &
\end{array}$$

So we find a unique specification for $t_{j+1,i}$ from $(j+1)$'s equation in (4.4):

$$t_{j+1,i} = \gamma_{j+1} - t_{j+1,i+1}w_{i+1} - \ldots - t_{j+1,l}w_l$$

(we showed that all values on the right-hand side are known from the previous steps).

Proceeding in this manner to the n's row, we determine the whole $\diamond$ element by element.

■

Remark: The construction of small pairwise independent sample spaces, presented here, is certainly not limited to binary strings ($S = \{0,1\}^n$). We can be easily extended to sequences over any field $\mathcal{F}$ (for example, over field $\mathcal{F} = \{0, \ldots, p-1\}$ with usual modular multiplication and addition, where p is a prime). We can simply replace 2-element field $\mathcal{F}_2 = \{0,1\}$ by an arbitrary field $\mathcal{F}$ (then $S = \mathcal{F}^n$), and all our arguments will remain valid (because they only depend on the distinction between zero and non-zero elements). The same remark refers to the first construction.

4.3 Hash Functions

4.3.1 Small pairwise independent sample spaces viewed as hash functions

Consider sequences of m elements over S as functions from $[m]$ to S. Then we can write our small sample space Ω as space of hash functions:

$$\Omega = \{h : [m] \rightarrow S\} = H$$

or, in our case (i.e. the second construction), $\Omega = \left\{h : \{0,1\}^l \rightarrow \{0,1\}^n\right\}$. We obtain hash functions (i.e. shrinking of input) when $l > n$.

Even in the first construction of small pairwise independent sample space, using procedure of expansion to the case $m > |S|$ (recall that the basic case was $m \leq |S|$), we get hash functions. If we have an embedding μ of large space S' (identified with a finite field), $|S'| \geq m$ into our small initial space S, and polynomial P over field of size m, we get hash function $h = \mu \circ P$ from S' to S.

This gives us the way of obtaining hash functions when we need k-wise independence, $k \neq 2$ (in this case the less problematic second construction doesn't work).

4.3.2 Good hash functions

Hash functions map long strings into short ones. We expect from hash functions a sort of "justice": each short string is hit by approximately the same number of long strings.

Definition 4.2 *Hash function h is called k-wise independent, if for any distinct $i_1, \ldots, i_k$ and any $\alpha_1, \ldots, \alpha_k$*

$$\Pr_{h \in_R H}\left[h(i_1) = \alpha_1 \wedge \cdots \wedge h(i_k) = \alpha_k\right] = \frac{1}{|S|^k}$$

In the first construction, the representation of k-wise independent hash function is of size $k \cdot l$. When considering $k = 2$ this representation is compatible to $l + 2n + 1$, obtained in our second construction.

Hash functions can be computed rather efficiently, in polynomial time or even better.

4.3.3 Applications of hash functions

Typically in applications we want to map large space $\{0,1\}^l$ into small space $\{0,1\}^n$ with hash functions, so that this mapping looks like random.

To define an arbitrary (boolean) hash function we need 2^l bits – to specify value at every point. To define a k-wise independent hash function we need only kl bits.

But usually we want even more: we have a subset $I \subset \{0,1\}^l$, and wish our mapping to be uniform also with respect to this subset I. If $|I| < 2^n$ we want that there will be no collisions, and even when I is bigger than the target set, we still want the mapping to distribute I evenly.

This case (I is a bit larger than $\{0,1\}^n$) often appears in complexity theory.

It turns out, that even pairwise independence gives us a considerable degree of uniformness in distribution, as states the following Lemma 4.3.1.

4.3.4 Hashing Lemma

In the following lemma, one may think of $m > 2^n$, although the Lemma holds also fo $m \leq 2^n$. Note that $\mu \stackrel{\text{def}}{=} \frac{|I|}{2^n}$ is the expected number of elements in I that are mapped by a random function (or uniformly chosen hash function) to any fixed point. Our focus is on $|I| > 2^n$ (i.e., $\mu > 1$), but the lemma holds also for $|I| \leq 2^n$ (i.e., $\mu \leq 1$).

Lemma 4.3.1 *Let $H = \{h : [m] \to \{0,1\}^n\}$ be a family of pairwise independent hash functions. Let $I \subseteq [m]$ and $\mu \stackrel{\text{def}}{=} \frac{|I|}{2^n}$.* THEN *for every $\alpha \in \{0,1\}^n$*

$$\Pr_{h \in H}\left[\Big| |\{i \in I \ : \ h(i) = \alpha\}| - \mu \Big| > \epsilon\mu\right] < \frac{1}{\epsilon^2 \mu}$$

Remarks: Recall that μ is the expected number of hits; and so the bound is meaningful when $\epsilon > \frac{1}{\sqrt{\mu}}$ (in which case it is less than 1). Setting $\epsilon = \mu^{-\frac{1}{3}}$, we obtain

$$\Pr_{h \in H}\left[\Big| |\{i \in I \ : \ h(i) = \alpha\}| - \mu \Big| > \mu^{\frac{2}{3}}\right] < \mu^{-\frac{1}{3}}$$

which is meaningful for $\mu > 1$.

Proof: The main thing in this proof is to define an appropriate random variable and makes some simple observations. After that the conclusion comes out naturally.

Fixing $\alpha \in \{0,1\}^n$, define for each $i \in I$ a random variable X_i:

$$X_i \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } h(i) = \alpha \\ 0 & \text{otherwise} \end{cases}$$

(the sample space here is H, and so X_i is actually $X_i(h)$).

Notice the following facts:

1. $\Pr[X_i = 1] = 2^{-n}$, because for fixed i and random $h \in H$ the value $h(i)$ is uniformly distributed in $\{0,1\}^n$. Therefore $\mathbf{E}(X_i) = 1 \cdot \Pr[X_i = 1] + 0 \cdot \Pr[X_i = 0] = 2^{-n}$.

2. The X_i's are pairwise independent (they are functions of pairwise independent values, i.e. the values $h(i)$'s).

3. $\text{VAR}(X_i) = \mathbf{E}(X_i^2) - \mathbf{E}(X_i)^2 = \mathbf{E}(X_i) \cdot (1 - \mathbf{E}(X_i)) \leq \mathbf{E}(X_i)$ *(valid for any* $\{0,1\}$*-valued random variables).*

4. For a random choice of h the number of elements from I that hit a particular element from $\{0,1\}^n$ equals $|\{i \in I \ : \ h(i) = \alpha\}| = \sum_{i \in I} X_i$.

5. $\mathbf{E}\left(\sum_{i \in I} X_i\right) = \sum_{i \in I} \mathbf{E}(X_i) = \frac{|I|}{2^n} = \mu$.

6. $\text{VAR}\left(\sum_{i \in I} X_i\right) = \sum_{i \in I} \text{VAR}(X_i) \leq \sum_{i \in I} \mathbf{E}(X_i) \leq \mu$.

These facts together with Chebishev's inequality (applied to $\sum_{i \in I} X_i$) imply:

$$\Pr\left[\left|\sum_{i \in S} X_i - \mathbf{E}\left(\sum_{i \in S} X_i\right)\right| > \epsilon\mu\right] < \frac{1}{(\epsilon\mu)^2} \cdot \text{VAR}\left(\sum_{i \in I} X_i\right) \leq \frac{\mu}{\epsilon^2\mu^2} = \frac{1}{\epsilon^2\mu}$$

and the lemma follows. ■

Oded's Note: *Using a k-wise independent family of hash functions allows to prove a stronger bound than the one provided by Lemma 4.3.1. Specifically, for such a family H and notations as in Lemma 4.3.1 the following holds:* for every $\alpha \in \{0,1\}^n$

$$\Pr_{h \in H}\left[\left| |\{i \in I \ : \ h(i) = \alpha\}| - \mu \right| > \epsilon\mu\right] < \left(\frac{k^2}{\epsilon^2\mu}\right)^{k/2}$$

The proof follows by using the stronger bounds provided in Lecture 2 for the sum of k-wise independent random variables. In particular, using the union bound and $k = n$, we get

$$\Pr_{h \in H}\left[\exists \alpha \in \{0,1\}^n \text{ s.t. } \left| |\{i \in I \ : \ h(i) = \alpha\}| - \mu \right| > \epsilon\mu\right] < \left(\frac{4k^2}{\epsilon^2\mu}\right)^{k/2}$$

which will be used in Lecture 6.

Oded's Note: *An alternative way of stating Lemma 4.3.1 follows. Let H, $I \subseteq [m]$ and $\mu = |I|/2^n$ be as in the lemma, and let X be uniformly distributed over I. Then, for every $\alpha \in \{0,1\}^n$, for all but an $\epsilon^{-2}\mu^{-1}$ fraction of the h's in H it holds that*

$$\left|\Pr\left[h(X) = \alpha\right] - \mu \cdot 2^{-n}\right| \leq \epsilon\mu \cdot 2^{-n}$$

where the probability is taken over the choice of X (i.e., uniformly over I). A different generalization of Lemma 4.3.1 is obtained by considering arbitrary random variables X over $[m]$ satisfying $\Pr[X = x] \leq \mu^{-1}2^{-n}$, for every x.

Bibliographic Notes

The constructions of pairwise independent sample spaces and hash functions (presented above) are due to Carter and Wegman [15]. Versions of the hashing lemma were discovered independently in [26] and [13]. The lemma can be viewed as an extension (or quantification) of the ideas underlying Sipser's work [40].

Lecture 5

Approximate Counting

Notes taken by Oded Lachish, Eran Ofek, Udi Wieder

Summary: This lecture discusses quantitative problems related to NP relations. Namely, given an element in an NP language, we ask how many witnesses this element has. We use Hash functions in order to prove that given an oracle to NP and an instance in an NP language, it is possible to approximate the number of witnesses for that instance.

5.1 Introduction

Given a set S, a few natural questions one can ask about S can be divided into two categories.

1. Qualitative Questions:
   - Decision Question - is S empty or not? (I.e., $S \stackrel{?}{=} \phi$)
   - Search Query - produce an element of S. Find y such that $y \in S$.
2. Quantitative Questions:
   - Counting - What is the cardinality of S. (I.e., $|S| =?$)
   - Uniform Generation - Generate an element of S with uniform probability among all possibilities.

Typically quantitative questions are harder to answer than qualitative ones. If one can count the number of elements in S then one can surely tell whether S is empty or not. We investigate what happens the other way around; namely, when being able to answer qualitative questions, what can be said about the ability to answer the quantitative ones?

5.1.1 Formal Setting

We are given a family of sets $\{S_x\}_{x\in\{0,1\}^*}$ where:

- $S_x \subseteq \{0,1\}^*$. Furthermore there exists a polynomial $p(\cdot)$ such that the length of elements in S_x is at most $p(|x|)$.
- Given x and y it is easy (=done in polynomial time) to check whether $y \in S_x$.

In other words, the family $\{S_x\}_{x\in\{0,1\}^*}$ defines an NP relation. Recall:

Definition 5.1 (NP relation) *: An NP relation is a relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$ such that:*

- *R is polynomial time decidable.*
- *There exists a polynomial $p(\cdot)$ such that for every $(x,y) \in R$, it holds that $|y| \leq p(|x|)$.*

We let S_x be the witness set of x (or *set of solutions for* x); i.e., $S_x \stackrel{\triangle}{=} \{y : (x,y) \in R\}$. The witness set of x (with respect to relation R) is also denoted by $R(x)$.

Using this notation, we rephrase the four problems at hand:

1. The Decision Question is the usual NP decision problem, namely given x check if $\exists y$ such that $(x,y) \in R$. In other words, check if $R(x) \stackrel{?}{=} \phi$.
2. The Search Problem is: given x find y such that $y \in R(x)$.
3. The Counting Problem: given x compute $|R(x)|$.
4. The Generation Problem: given x, generate y uniformly in $R(x)$.

5.1.2 How close are the quantitative questions to the qualitative ones?

First we investigate the connection between the two qualitative questions. How close are the Decision Problem and the Search Problem? Surly, given an oracle to the Search Problem, it is easy to solve the Decision Problem. A well known result shows that for NP-complete relations the other way around holds as well. In other words:

Fact 5.1.1 (Self Reducibility of NP): *For any NP-relation R, given an element x and an oracle access to NP decision questions, it is possible in polynomial time to find y such that $y \in R(x)$.*

Note that the oracle can answer any query which is an NP decision problem, and not necessarily queries regarding the specific relation R. In case R is NP-complete, we can use an oracle to $L(R) \stackrel{\triangle}{=} \{x : R(x) \neq \phi\}$ instead of an oracle to NP.

Fact 5.1.1 demonstrates a tight connection between the two qualitative questions. In the rest of the lecture we show that in some sense the quantitative questions are not much harder than the qualitative ones. Namely, given an oracle to the Decision Question, it is possible to approximate the number of witnesses of an element. In the next lecture we will present a result in the spirit of Fact 5.1.1 that demonstrates a tight connection between counting and uniform generation.

5.2 Approximating the size of of the witness set with oracle access to NP

Goal: For a fixed NP-relation R, provide an procedure that, given an element x, an approximation paramater $\epsilon > 0$, and oracle access to NP, runs in time *poly* $\left(\frac{|x|}{\epsilon}\right)$ and satisfies:

$$\Pr\left[Alg^{NP}(x) \notin (1 \pm \epsilon)|R(x)| \leq \frac{1}{3}\right].$$

As shown in the next section, the error probability can be reduced at a cost logarithmic in the desired error bound.

Tools: The tools we use are a family of pairwise independent Hash functions $H_{n,m}$, that hashes $\{0,1\}^n \to \{0,1\}^m$ $(m < n)$, and an oracle for any NP-complete language.

The strategy for solving the problem is first to show how to find a solution to the case in which $|R(x)|$ is small. (This is done using an oracle to NP). Then show how to reduce the general case (a large witness set), to a case in which the witness set is small. (This is done using the hash functions). Using a family of pairwise-independent hash functions has a price: there is a constant probability that the hash function chosen does not behave well, we denote this probability by δ. When the chosen hash function behaves well, the answer of the algorithm will be in the interval $(1 \pm \epsilon) \cdot |R(x)|$. Where ϵ is called the deviation parameter, and δ is called the error probability. Thus, for an NP relation R and an input x, the algorithm we describe efficiently produces a result that behaves as follows:

$$Prob\left[Alg^{\mathcal{NP}}(x) \notin (1 \pm \epsilon)|R(x)|\right] < \delta$$

where the probability is over the coin tosses of the algorithm.

5.2.1 The algorithmic scheme

Assume that $|x| = n$ and that $R(x) \subseteq \{0,1\}^n$ (this assumption is with out loss of generality, since padding can bring the element x and the witnesses to the same size). Let $l = O(\lg n)$.

Following is a sketch of the algorithm (which is actually an algorithmic scheme):

Algorithm A:

1. If $|R(x)| < 2^l$ output $|R(x)|$.
2. If $|R(x)| \geq 2^l$ then
   For $i = 1$ to $n - l$
   - Select $h \in_R H_{n,i}$.
   - If $|R(x) \cap h^{-1}(0^i)| < 2^l$ then
     output $2^i \cdot |R(x) \cap h^{-1}(0^i)|$ and Halt.

In Stage 1 we handle the case in which $|R(x)|$ is small. In the iterations of Stage 2 we hash the witness set into an increasingly larger set, until the number of witnesses that are mapped to 0^i is small. Specifically, if the hash function maps $\{0,1\}^n$ to $\{0,1\}^i$, we expect it to map $\frac{|R(x)|}{2^i}$ witnesses to each element in $\{0,1\}^i$. Once 2^i is the 'right' number, we have that a small number of elements are mapped into 0^i. (0^i was chosen arbitrarily). This case is similar to the case of a small a witness set, so is identifiable, and can be counted exactly. As a result the size of the set of witnesses mapped to 0 is used to approximate the entire number of witnesses.

5.2.2 Implementation of the algorithmic scheme

How to perform Stage (1): We check if $|R(x)| < 2^l$ by defining the following NP relation:

$$R' = \{x, (y_1, \ldots, y_{2^l}) : y_1 \neq \ldots \neq y_{2^l} \text{ and } (x, y_1), \ldots (x, y_{2^l}) \in R\}.$$

That is, $L(R') = \{x : \exists y'(x, y') \in R'\}$ contains x iff x has at least 2^l *distinct* witnesses (i.e. $|R(x)| \geq 2^l$). Since R is an NP relation we know that the length of each y_i is polynomially bounded by the length of x. Since 2^l is polynomial in $|x|$ we have that the length of $(y_1, y_2, \ldots, y_{2^l})$ is also polynomially bounded by $|x|$. We conclude that R' is an NP relation.

Thus, to check whether $|R(x)| \geq 2^l$ we query the oracle whether $R'(x) \stackrel{?}{=} \phi$. If the oracle answers 'NO' then $|R(x)| < 2^l$. To compute the exact size of $|R(x)|$ define the following NP relation:

$$R'' = \{(x, 1^t), (y_1, \ldots, y_t) : y_1 \neq \ldots \neq y_t \text{ and } (x, y_1), \ldots (x, y_t) \in R\}.$$

That is, the language $L(R'')$ contains $(x, 1^t)$ iff x has at least t *distinct* witnesses (i.e. $|R(x)| \geq t$). In order to ensure that R'' is indeed an NP relation, we have to ensure that the length of the witnesses is polynomially bounded by the length of the instances. Note that t is represented in unary, therefore the length of the instance $(x, 1^t)$ is $|x| + t$. The witnesses are composed of t y_i's, whose lengths are polynomially bounded by $|x|$. Therefore, for every t, the length of $(y_1, y_2, \ldots, y_t)$ is polynomially bounded by $|(x, 1^t)|$ and we conclude that R'' is an NP relation. Thus, to find the exact size of $R(x)$, we iteratively query the oracle on the decision problem of $R''(x, 1^t)$ for $t = 1 \ldots 2^l$ and find the largest t for which $(x, 1^t) \in L(R'')$. Note that 2^l is polynomial in n, thus the whole operation runs in polynomial time.

How to perform stage (2)**:** For each iteration i define the following NP relation:

$$R''' = \{(x, h), (y_1, \ldots, y_{2^l}) : y_1 \neq \ldots \neq y_t \text{ and } h(y_1) = h(y_2) = \cdots = h(y_{2^l}) = 0^i$$
$$\text{and } (x, y_1), \ldots, (x, y_{2^l}) \in R\}$$

That is, $L(R''')$ contains (x, h) iff x is in $L(R)$, and it has at least 2^l distinct witnesses mapped to 0^i by h. Note that the hash function can be evaluated in time that is polynomial in $|x|$. Therefore it is possible to verify in polynomial time whether $((x, h), (y_1, y_2, \ldots, y_{2^l})) \in R'''$. Thus $L(R''') \in \mathcal{NP}$. Furthermore, since h can be represented in space which is polynomial in $|x|$, the length of (x, h) is polynomial in $|x|$, and it is possible to produce an NP query for R''' in polynomial time (in $|x|$). Querying the oracle about $R'''(x)$ enables us to identify when less than 2^l distinct witnesses were mapped to 0^i by h. In this case, finding the exact number of such witnesses, is done in a similar manner to the first case when $|R(x)|$ was small.

5.2.3 Analysis of the algorithm

The parameter l is chosen to be big enough so that $\epsilon \geq 10 \cdot 2^{-l/2}$ and also $2^l \geq n^2$. Note that since the running time is polynomial in 2^l, it follows that the running time is polynomial in $\max\{\text{n}, \frac{1}{\epsilon}\}$. This means that ϵ can be at best small as $\frac{1}{poly(n)}$, otherwise the running time would not be polynomial in n.
Let t be such that $2^t = |R(x)|$ (for sake of simplicity and without loss of generality assume t is an integer).

General strategy of the proof:
We will show that the algorithm is not likely to terminate before iteration $t - l$; that is, with high probability it will reach iteration $t - l$. Then, we show that given that A reached iteration $t - l + 1$ it is very likely that A will terminate in this iteration. This implies that the algorithm is likely to terminate in one of the iterations $\{t - l, t - l + 1\}$. The last step is to show that in both iterations ($i = t - l$ or $i = t - l + 1$) the value $|R(x) \cap h^{-1}(0^i)| \cdot 2^i$ is an ϵ approximation of $|R(x)|$ with high probability. Thus if A terminates in these iterations (as expected) it will output an ϵ approximation of $|R(x)|$ with high probability.

Recall the hashing lemma from the previous lecture:

Lemma 5.2.1 (Hashing lemma) *Let $H = \{h : \{0,1\}^n \rightarrow \{0,1\}^i\}$ be a family of pairwise independent hash functions. Let $I \subseteq \{0,1\}^n$ and $\alpha \in \{0,1\}^i$,and let $\rho = \frac{|I|}{2^i}$. Then*

$$\Pr_h \left[\left| |I \cap h^{-1}(\alpha)| - \rho \right| \geq \epsilon \cdot \rho \right] \leq \frac{1}{\epsilon^2 \cdot \rho}$$

where h is uniformly selected in H.

We will use this lemma to prove that algorithm A will return a good approximation of $|R(x)|$, with high probability. We do this by examining the value $|R(x) \cap h^{-1}(0^i)|$ in various iterations of A. Let us fix some i (iteration number in the running of A).

Recall that since $h \in H$ is pairwise independent, it follows that

$$E_h \left[|R(x) \cap h^{-1}(0^i)| \right] = \frac{|R(x)|}{2^i} \tag{5.1}$$

We denote $\rho \stackrel{\text{def}}{=} \frac{|R(x)|}{2^i}$. For every $y \in R(x)$ denote by X_y the following random variable:

$$X_y = X_y(h) = \begin{cases} 1 & h(y) = 0^i, \\ 0 & otherwise. \end{cases}$$

where the probability space is of all choices of $h \in H$.
Recall that by the properties of H, in iteration i we have $\Pr_h[X_y = 1] = \frac{1}{2^i}$. Notice that

$$|R(x) \cap h^{-1}(0^i)| = \sum_{y \in R(x)} X_y$$

Thus, by linearity of the expectation:

$$E_h \left[|R(x) \cap h^{-1}(0^i)| \right] = \sum_{y \in R(x)} E_h(X_y)$$

$$= \sum_{y \in R(x)} \Pr[X_y = 1] = \frac{|R(x)|}{2^i} \quad = \quad \rho$$

which indeed agrees with equation (5.1).

First we show that the algorithm A is not likely to terminate in any of the iterations $i \in \{1, 2, .., t-l-1\}$.

Claim 5.2.2 *For every $i \in \{1, 2, .., t-l-1\}$ the probability that A terminates at iteration i is smaller than $\frac{1}{10n}$.*

Proof:

$$\rho = \frac{|R(x)|}{2^i} \geq \frac{2^t}{2^{t-l-1}} \geq 2^{l+1}$$

We wish to bound the probability that algorithm A terminates at iteration i, so we need to bound the probability that $|R(x) \cap h^{-1}(0^i)| < 2^l$. Notice that if $|R(x) \cap h^{-1}(0^i)| < 2^l$ then $||R(x) \cap h^{-1}(0^i)| - \rho| > \frac{1}{2} \cdot \rho$ (this is true since $\rho \geq 2^{l+1}$). By using the hashing lemma we can bound the probability that $||R(x) \cap h^{-1}(0^i)| - \rho| > \frac{1}{2} \cdot \rho$:

(The set $R(x) \subseteq \{0,1\}^n$ plays the roll of the set I from the lemma).

$$\Pr_h \left[||R(x) \cap h^{-1}(0^i)| - \rho| > \frac{1}{2} \cdot \rho \right] \leq \frac{1}{(\frac{1}{2})^2 \cdot \rho}$$

$$\leq \frac{1}{(\frac{1}{2})^2 \cdot 2^{l+1}} = 2^{-l+1} \leq 2 \cdot \frac{1}{n^2} \leq \frac{1}{10 \cdot n}$$

(for big enough n's). ■

Now we will show that A is not likely to continue beyond iteration $t - l + 1$.

Claim 5.2.3 *Given that A reached iteration $t - l + 1$, the probability that A terminates at this iteration is at least* 0.9.

Proof: Here, we have

$$\rho = \frac{|R(x)|}{2^{t-l+1}} = \frac{2^t}{2^{t-l+1}} = 2^{l-1}$$

We wish to bound the probability that algorithm A does not terminate in this iteration. This happens only if $|R(x) \cap h^{-1}(0^i)| \geq 2^l$, but this event is contained in the event that $||R(x) \cap h^{-1}(0^i)| - \rho| \geq \rho$ (since $\rho = 2^{l-1}$). We use the hashing lemma to bound this probability:

$$\Pr_h \left[|R(x) \cap h^{-1}(0^i)| - \rho| \right] \geq 1 \cdot \rho] \leq \frac{1}{\rho}$$

$$= \frac{1}{2^{l-1}} = 2^{-l+1} = 2 \cdot \frac{1}{n^2} \leq 0.1$$

(for large enough n's) ■

It remains to show that given that A terminates at iteration $t - l$ or $t - l + 1$ it is likely that A outputs an ϵ approximation of $R(x)$.

Claim 5.2.4 *Assume that A terminates at iteration $t - l$ or $t - l + 1$, then*

$$\Pr_h [A(x) \notin (1 \pm \epsilon) \cdot |R(x)|] \leq \frac{1}{10}$$

Proof: Here, we have

$$\rho = \frac{|R(x)|}{2^i} \geq \frac{2^t}{2^{t-l+1}} = 2^{l-1}$$

Since $\frac{A(x)}{2^i} = |R(x) \cap h^{-1}(0^i)|$, it follows that the following two conditions are equivalent:

$$|A(x) - |R(x)|| \geq \epsilon \cdot |R(x)| \tag{5.2}$$

$$\left| |R(x) \cap h^{-1}(0^i)| - \frac{|R(x)|}{2^i} \right| \geq \epsilon \cdot \frac{|R(x)|}{2^i} \tag{5.3}$$

Equation (5.3) can be written as:

$$\left| |R(x) \cap h^{-1}(0^i)| - \rho \right| \geq \epsilon \cdot \rho$$

Using the hashing lemma, we have

$$\begin{aligned}\Pr_h\left[\left||R(x)\cap h^{-1}(0^i)|-\rho\right|\geq\epsilon\cdot\rho\right] &\leq \frac{1}{\epsilon^2\cdot\rho}\\ &\leq \frac{1}{\epsilon^2\cdot 2^{l-1}}\\ &\leq \frac{1}{(10\cdot 2^{l/2})^2\cdot 2^{l-1}}\leq\frac{1}{10}\end{aligned}$$

Using the equivalence of equations (5.2) and (5.3), the claim follows. ∎

Corollary 5.2.5 *The probability that A outputs an ϵ approximation of $|R(x)|$ is at least $\frac{2}{3}$.*

Proof: The probability that A terminates before iteration $t-l$ is at most $\frac{1}{10}$ (using the union bound with Claim 1). It follows that the probability for algorithm A to terminate at iteration $t-l$ or at iteration $t-l+1$ is at least 0.8, because

$$\Pr[\text{A terminates at iteration t-l or t-l+1}] =$$

$$\Pr\left[\begin{array}{l}\text{A reaches,}\\ \text{iteration t-l}\end{array}\right]\cdot\Pr\left[\begin{array}{l|l}\text{A terminates at,} & \text{A reaches,}\\ \text{iteration t-l or t-l+1} & \text{iteration t-l}\end{array}\right]$$

$$\geq 0.9\cdot 0.9 > 0.8$$

Using again the union bound with Claim 3 we conclude that the probability that A terminates at iterations $t-l$ or $t-l+1$ and outputs an ϵ approximation is at least $0.8-0.1>\frac{2}{3}$. ∎

5.3 Amplifying the success probability

In the previous section we saw that we can approximate the size of the witness set with deviation ϵ, and in running time that is polynomial in $\frac{1}{\epsilon}$. The success probability of the approximation was shown to be larger than $\frac{2}{3}$. Like in many other probabilistic algorithms, we wish to amplify the success probability. We will show that in polynomial time, it is possible to reduce the error probability to be exponentially small.

We start tackling this problem in a standard way: assume we wish the error probability to be smaller than δ. We set $t\stackrel{\triangle}{=}O(\log(\frac{1}{\delta}))$. The algorithm A is invoked t times, each invocation with fresh random coins, and we receive t (possibly different) answers. The question is what to do with these t answers?

At first thought one might think of taking the average of these results. On second thought this idea should be excluded, since the wrong solutions might be very wrong. If it happens that the algorithm miserably fails, it might give an answer that is so far from the true one, that it would skew the average. Consider the following illustrative example: Assume $|R(x)|=2^{\frac{n}{2}}$, and $\Pr[A(x)=2^{\frac{n}{2}}]=\frac{2}{3}$, yet $\Pr[A(x)=2^n]=\frac{1}{3}$. That is, even though A is correct most of time, if it fails it fails miserably. Now the average is $E\left(A(x)\right)=\frac{2}{3}\cdot 2^{\frac{n}{2}}+\frac{1}{3}\cdot 2^n=\Theta(2^n)$, which is very far from the correct answer.

A much better idea is to use the median of the t results. Let A' be the following algorithm:

Algorithm A':

1. Set $t \stackrel{\triangle}{=} O(\log(\frac{1}{\delta}))$
2. Invoke the algorithm A, t times, each invocation with fresh random coins, and receive t answers.
3. Output the median of the t answers.

Claim 5.3.1 *Prob* $[A'(x) \notin (1 \pm \epsilon) \cdot |R(x)|] \leq \delta$

Proof: We use the following fact:

Fact 5.3.2 *If the median of t numbers is outside the range (a,b) then at least $\frac{t}{2}$ of the numbers are outside the range (a,b).*

Define random variables

$$X_i = \begin{cases} 1 & \text{the } i\text{'th invocation of A gives a value in } (1 \pm \epsilon) \cdot |R(x)|, \\ 0 & \textit{otherwise.} \end{cases}$$

Note that $\Pr[X_i = 1] \geq \frac{2}{3}$, therefore $E(\sum_{i=1}^{t} X_i) \geq \frac{2t}{3}$.
According to Fact 5.3.2, if A' fails then at least half of the invocations of A failed. This means that (in case of failure) $\sum_{i=1}^{t} X_i \leq \frac{t}{2}$ and thus the sum is very far from it's expectation. We use the Chernoff bound to show that this is very unlikely.
According to Fact 5.3.2 we have:

$$\begin{aligned} \Pr[A'(x) \notin (1 \pm \epsilon) \cdot |R(x)|] &\leq \Pr\left[\sum_{i=1}^{t} X_i \leq \frac{t}{2}\right] \\ &\leq \Pr\left[\left|\sum_{i=1}^{t} X_i - E(\sum_{i=1}^{t} X_i)\right| \geq \frac{t}{6}\right] \\ &\leq \exp\left(-\left(\frac{1}{6}\right)^2 \cdot t\right) \end{aligned}$$

where the last inequality is due to Chernoff Bound.
Thus, setting $t = 36 \ln\left(\frac{1}{\delta}\right)$ we get $\Pr[A'(x) \notin (1 \pm \epsilon) \cdot |R(x)|] \leq \delta$, and the claim follows. ■

Bibliographic Notes

The approximate counting (oracle) procedure presented above is a variant of a procedure originally suggested by Stockmeyer [42].

Lecture 6

Uniform Generation

Notes taken by Eden Chlamtac and Shimon Kogan

Summary: This lecture continues the discussion of approximate counting and uniform generation using an NP-oracle begun in the previous lecture, where we saw an approximate counting algorithm. Here we will see the equivalence (up to Turing reductions) of approximate counting to uniform generation of NP-witnesses. Together with the approximate counting algorithm from last lecture, this yields a uniform generation algorithm using an NP-oracle. We conclude with an alternative (direct) uniform generation algorithm, which uses n-wise independent hash functions (as opposed to the pairwise independent hash functions used for approximate counting).

6.1 Introduction

Recall from the previous lecture that we were concerned with answering quantitative questions about sets for which we have an oracle. Specifically, given some NP-relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$, and some input string $x \in \{0,1\}^n$, we would like to answer quantitative questions regarding the set

$$S = R(x) \stackrel{\text{def}}{=} \{y : (x,y) \in R\} \subseteq \{0,1\}^n$$

(Note that we assume if $y \in R(x)$ then $|y| = n = |x|$. This is without loss of generality— it can be achieved with suitable padding.)

The previous lecture was concerned with *approximate counting*—approximating the size of S. In this lecture we are concerned with the closely related problem of *uniform generation*, or sampling with uniform distribution an element in S. In fact, we will show that the two computational problems are equivalent. That is, we show that *Approximate Counting and Uniform Generation are computationally equivalent.*

6.2 Approximate Counting Reduces to Uniform Generation

At our current state of knowledge (i.e., having seen n approximate counting algorithm in the previous lecture but not a uniform generation one), this is the less interesting direction of the equivalence. Still, it is simpler to explain and is needed to establish the computational equivalence of the two tasks.

6.2.1 Intuition

Assume we have a generic uniform generation algorithm UniGen, which for any NP-relation R and input x outputs $\mathrm{UniGen}_R(x)$—a string chosen with uniform distribution from the set $R(x)$.

Using UniGen we will generate a sufficiently large sample from $|S|$ to estimate what fraction of the strings in $|S|$ start with a particular bit σ. Denoting this subset of S by S_σ, we estimate

$$p_\sigma \stackrel{\text{def}}{=} \frac{|S_\sigma|}{|S|}$$

We then recursively approximate the quantity $|S_\sigma|$, and then produce an approximation for $|S|$ using the formula

$$|S| = \frac{1}{p_\sigma} \cdot |S_\sigma|$$

The recursion will proceed to S_σ such that $p_\sigma > \frac{1}{2}$ (the larger of the two values p_σ and $p_{\bar{\sigma}}$), since we need a lower bound on p_σ in order to get a good bound on an estimate of the form $\frac{1}{p_\sigma \pm \epsilon}$.

Our goal is to define an algorithm Approx, which, given some arbitrarily small parameters $\epsilon > 0$, $\delta > 0$, satisfies:

$$\Pr[\mathrm{Approx}_R(x) \notin (1 \pm \epsilon) \cdot |R(x)|] < \delta$$

6.2.2 Formal Definition

Recall that our uniform generation algorithm relates to the NP-relation R. In order to apply it to subsets of S we need to define new NP-relations. Formally, we define

$$R' \stackrel{\text{def}}{=} \{((x, \alpha), \beta) : \alpha, \beta \in \{0,1\}^*, (x, \alpha \circ \beta) \in R\}$$

(Here, $\alpha \circ \beta$ denotes the concatenation of the two strings α and β.) Clearly, if R is an NP-relation, then R' is also an NP-relation. Moreover, for any $\alpha_0 \in \{0,1\}^*$, R' satisfies

$$R'(x, \alpha_0) = \{\beta : (x, \alpha_0 \circ \beta) \in R\} \cong \{\alpha \circ \beta \in R(x) : \alpha = \alpha_0\}$$

Note that the set on the right is of the sort we want to quantify in the recursion.

Note that since the error and deviation bounds will increase with successive approximations at various levels of recursion, we will have to start out with stricter bounds. Specifically, we take

$$\tilde{\epsilon} = \tfrac{\epsilon}{3n} \text{ and } \tilde{\delta} = \tfrac{\delta}{n}$$

(Here ϵ and δ are the deviation and error parameters in the performance specification noted above).

Since $|R(x)| = |R'(x, \lambda)|$ (where λ is the empty string), for the sake of uniformity of notation, we present an algorithm for approximating $|R'(x, \alpha)|$ where the string $\alpha \in \{0,1\}^*$ is part of the input (along with $x \in \{0,1\}^n$ of course), and initially $\alpha = \lambda$.

6.2.3 Algorithm $\mathrm{Approx}_{R'}(x, \alpha)$

1. We let $n \stackrel{\text{def}}{=} |x|$ as before. If $|\alpha| = n$ return $\begin{cases} 1 & \text{if } \alpha \in R(x) \\ 0 & \text{otherwise} \end{cases}$

2. Otherwise, for $i = 1, \ldots, m$ uniformly and independently generate sample points in $R'(x, \alpha)$:
$$s_i \leftarrow \mathrm{UniGen}_{R'}(x, \alpha)$$
where
$$m = O\left(\frac{1}{\tilde{\epsilon}^2} \log \frac{1}{\tilde{\delta}}\right)$$

3. For $\sigma = 0, 1$ set
$$\tilde{p}_\sigma \stackrel{\text{def}}{=} \frac{|\{i : s_i \in R'(x, \alpha \circ \sigma)\}|}{m}$$
(an approximation for $p_\sigma \stackrel{\text{def}}{=} \frac{|R'(x,\alpha\circ\sigma)|}{|R'(x,\alpha)|}$)

4. For σ s.t. $\tilde{p}_\sigma \geq \tilde{p}_{\bar{\sigma}}$ return
$$\frac{1}{\tilde{p}_\sigma} \cdot \mathrm{Approx}_{R'}(x, \alpha \circ \sigma)$$

6.2.4 Analysis

We would like to see how the error and deviation bounds propagate through the recursion. First note that in step (2), at every level of the recursion, by choice of m and applying Chernoff bound, we get:
$$\Pr[|p_\sigma - \tilde{p}_\sigma| > \tilde{\epsilon}] < \tilde{\delta}$$
At the ith level of recursion (where $|\alpha| = i$) we will denote the error and deviation bounds by δ_i and ϵ_i, so that:
$$\Pr[\mathrm{Approx}_{R'}(x, \alpha) \notin (1 \pm \epsilon_i) \cdot |R'(x, \alpha)|] < \delta_i$$
Given these two probability bounds, let us show how the error and deviation propagate from level $i + 1$ of the recursion up to level i. Recall that:
$$\mathrm{Approx}_{R'}(x, \alpha) \stackrel{\text{def}}{=} \frac{\mathrm{Approx}_{R'}(x, \alpha \circ \sigma)}{\tilde{p}_\sigma}$$
We know that with probability $> 1 - (\tilde{\delta} + \delta_{i+1})$ we have both $|p_\sigma - \tilde{p}_\sigma| \leq \tilde{\epsilon}$ and
$$\mathrm{Approx}_{R'}(x, \alpha \circ \sigma) \in (1 \pm \epsilon_{i+1}) \cdot |R'(x, \alpha)|$$
In this case it holds that:
$$\begin{aligned}
\frac{\mathrm{Approx}_{R'}(x, \alpha \circ \sigma)}{\tilde{p}_\sigma} &= \frac{1 \pm \epsilon_{i+1}}{\tilde{p}_\sigma} \cdot |R'(x, \alpha \circ \sigma)| \\
&= \frac{1 \pm \epsilon_{i+1}}{\tilde{p}_\sigma} \cdot p_\sigma \cdot |R'(x, \alpha)| \\
&= \frac{(\tilde{p}_\sigma \pm \tilde{\epsilon}) \cdot (1 \pm \epsilon_{i+1})}{\tilde{p}_\sigma} \cdot |R'(x, \alpha)| \\
&= \left((1 \pm \epsilon_{i+1}) \pm \frac{\tilde{\epsilon}}{\tilde{p}_\sigma}(1 \pm \epsilon_{i+1})\right) \cdot |R'(x, \alpha)| \\
&= \left(1 \pm \left(\epsilon_{i+1} + \frac{\tilde{\epsilon}}{\tilde{p}_\sigma}(1 + \epsilon_{i+1})\right)\right) \cdot |R'(x, \alpha)|
\end{aligned}$$

where the second equality follows from from the equality $|R'(x, \alpha \circ \sigma)| = p_\sigma |R'(x, \alpha)|$, which follows immediately from the definition of p_σ. Now it remains to bound

$$\epsilon_i \stackrel{\text{def}}{=} \epsilon_{i+1} + \frac{\tilde{\epsilon}}{\tilde{p}_\sigma}(1 + \epsilon_{i+1})$$

Recall that by our choice of $\tilde{p}_\sigma$, we know

$$\tilde{p}_\sigma \geq 1/2$$

Also, as will be apparent from the following analysis, if we assume w.l.o.g. that $\epsilon < 1/2$ then it is easy to show that

$$\epsilon_i < 1/2$$

for all i.
Keeping these two inequalities in mind, we have

$$\begin{aligned}
\epsilon_i &= \epsilon_{i+1} + \frac{\tilde{\epsilon}}{\tilde{p}_\sigma}(1 + \epsilon_{i+1}) \\
&< \epsilon_{i+1} + \frac{\tilde{\epsilon}}{1/2}(1 + (1/2)) \\
&= \epsilon_{i+1} + 3\tilde{\epsilon}
\end{aligned}$$

So clearly the bounds we chose—

$$\tilde{\epsilon} = \tfrac{\epsilon}{3n} \text{ and } \tilde{\delta} = \tfrac{\delta}{n}$$

are sufficient.

6.2.5 An alternative reduction of Approximate Counting to Uniform Generation

Here is another way of reducing approximate counting to uniform generation, suggested by Udi Wieder. Rather than recursively taking smaller and smaller subsets of S, we may take some predefined set K (with known cardinality and easily decidable) of comparable size to S, such that $S \bigcap K = \emptyset$, and then get a good approximation of S by using repeatedly generating random samples in $S \cup K$, to approximate

$$\beta \stackrel{\text{def}}{=} \frac{|S|}{|S \bigcup K|}$$

Since $|K|$ is known, this will yield an approximation of $|S|$ derived from the following equation:

$$|S| = |K| \cdot \frac{\beta}{1 - \beta}$$

Here we will want to be able to bound β both in terms of upper- and lower-bounds. The situation is similar to that which we had with p_σ in the last algorithm. If we had $\beta \approx 0$ or $\beta \approx 1$ then knowing we had a good approximation of the form $\beta \pm \epsilon$ would not help, since even a small deviation will no longer yield a good relative approximation for β or $\frac{1}{1-\beta}$. Hence, it is essential that we find a set K of comparable size to S (where $S \subseteq \{0,1\}^n$).

Specifically, for all $i = 1, \ldots, n$ we define

$$K_i \stackrel{\text{def}}{=} \{1^{n+1} \circ y : y \in \{0,1\}^i\} \subseteq \{0,1\}^{n+i+1}$$

$$S_i \stackrel{\text{def}}{=} \{0^{i+1} \circ s : s \in S\} \subseteq \{0,1\}^{n+i+1}$$

Note that these sets satisfy the following properties for all i:

- $|K_i| = 2^i$
- $|S_i| = |S|$
- $K_i \cap S_i = \emptyset$ (since $z \in K_i$ start with 1, and $z \in S_i$ start with 0)

Intuitively, $S_i \cup K_i$ is a witness set for the input $x \circ 01^i$. This motivates the definition of the following NP-relation:

$$R'' \stackrel{\text{def}}{=} \left\{ (x \circ 01^i, z) : \begin{array}{l} 1 \leq i \leq |x| \wedge x \in \{0,1\}^* \\ \wedge \left[(z = 0^{i+1} \circ y \wedge (x,y) \in R) \vee ((z = 1^{n+1} \circ y) \wedge y \in \{0,1\}^i) \right] \end{array} \right\}$$

Clearly, for some $1 \leq i \leq n$ we have $\frac{1}{2}|K_i| \leq |S| \leq |K_i|$ (where $S = R(x)$ and $x \in \{0,1\}^n$ as above). If β is defined as above for $K = K_i$ (i.e. $\beta = \frac{|S|}{|S|+|K_i|}$), then we get $\frac{1}{3} \leq \beta \leq \frac{1}{2}$. Hence if we can find such an i, knowing β is bounded as above, we can guarantee a good approximation of $|S|$. Here, since the deviation does not need to propagate through any recursion (there is no recursion), we can take $\tilde{\epsilon} \stackrel{\text{def}}{=} \frac{\epsilon}{9}$. The error remains as in the last algorithm (we still apply union bound on the probabilities), i.e. $\tilde{\delta} \stackrel{\text{def}}{=} \frac{\delta}{n}$. Here's the formal algorithm:

1. For $i = 1$ to n do:
   - For $j = 1, \ldots, m$ uniformly and independently generate sample points in $S \bigcup K_i$:
   $$s_j \leftarrow \text{UniGen}_{R''}(x \circ 01^i)$$
   where again
   $$m = O\left(\frac{1}{\tilde{\epsilon}^2} \log \frac{1}{\tilde{\delta}}\right)$$
   - Set $\beta_i \stackrel{\text{def}}{=} \frac{|\{j : s_j \in S_i\}|}{m}$
2. For $i_0 \stackrel{\text{def}}{=} \min\left\{i : \beta_i \geq \frac{1}{4}\right\}$ return
$$2^{i_0} \cdot \frac{\beta_{i_0}}{1 - \beta_{i_0}}$$

6.3 Uniform Generation Reduces to Approximate Counting

Given our current state of knowledge, this is indeed the more interesting reduction.

6.3.1 Intuition

Suppose we had perfect counting. In particular, suppose we could calculate the exact quantity $|S|$, and for $\sigma = 0, 1$ we could calculate the size of the set of all elements in S which start with σ (denoted S_σ), as in the previous section. In particular, we could compute the exact values

$$p_\sigma \stackrel{\text{def}}{=} \frac{|S_\sigma|}{|S|}$$

We want to output $y \in S$ with uniform distribution. So

- with probability p_0 output (recursively) $y \in S_0$ (with uniform distribution)
- with probability p_1 output (recursively) $y \in S_1$ (with uniform distribution)

6.3.2 Formal Definitions

Now assume we have an approximate counting algorithm Approx which for any NP-relation R and inputs x, ϵ, δ returns $\text{Approx}_R(x, \epsilon, \delta)$ where

$$\Pr[\text{Approx}_R(x, \delta, \epsilon) \notin (1 \pm \epsilon)|R(x)|] < \delta$$

and

$$\text{Approx}_R(x, \delta, \epsilon) = |R(x)| \text{ if } |R(x)| = 0$$

Now using Approx we'll generate a random element of $R(x)$ bit by bit in a recursive manner. This we do once more using the modified form of R which we defined in the previous section:

$$R' \stackrel{\text{def}}{=} \{((x, \alpha), \beta) : \alpha, \beta \in \{0,1\}^*, (x, \alpha \circ \beta) \in R\}$$

where we assume that $\Pr[\text{Approx}_{R'}(x, \alpha) \notin (1 \pm \epsilon)|R'(x, \alpha)|] < \delta$ and $\text{Approx}_{R'}(x, \alpha) = |R'(x, \alpha)|$ if $|R'(x, \alpha)| = 0$. First we'll define an auxiliary algorithm ShApprox which returns slightly shifted approximation values. Then using this algorithm we'll define the uniform generation algorithm UniGen. As always we assume that in R the length of all witnesses is equal to the length of the input x.

6.3.3 Algorithm $\text{ShApprox}_{R'}(x, \alpha)$

1. Let $n = |x|, i = |\alpha|$
2. Return $\begin{cases} \frac{4n^2+4n-4i}{4n^2+1} \cdot \text{Approx}_{R'}(x, \alpha) & if \quad i < n \\ \text{Round}(\text{Approx}_{R'}(x, \alpha)) & if \quad i = n \end{cases}$

In all the calls made to Approx by this algorithm we fix ϵ and δ to be $\epsilon = \frac{1}{4n^2}$ and $\delta = \frac{1}{3n^2}$. Notice that by the above definition we have that for $i = n$ ShApprox returns 1 if $\alpha \in R(x)$ and 0 otherwise. The purpose of this shifted approximation (as we'll show later in the analysis) is to ensure that with high probability we have:

$$\text{ShApprox}_{R'}(x, \alpha \circ 0) + \text{ShApprox}_{R'}(x, \alpha \circ 1) \leq \text{ShApprox}_{R'}(x, \alpha)$$

6.3.4 Algorithm $\text{UniGen}_{R'}(x, \alpha)$

1. Let $n = |x|, i = |\alpha|$
2. If $i = n$ return λ (where λ is the empty string)
3. For $\sigma = 0, 1$ set

$$\tilde{p}_\sigma \stackrel{\text{def}}{=} \frac{\text{ShApprox}_{R'}(x, \alpha \circ \sigma)}{\text{ShApprox}_{R'}(x, \alpha)}$$

   In this calculation, reuse value returned by calculation of $\text{ShApprox}_{R'}(x, \alpha)$ in the previous level of recursion (unless this is the first level of recursion, in which case we apply $\text{ShApprox}_{R'}(x, \lambda)$ once and use the same value in calculating p_0 and p_1).
4. If $\tilde{p}_0 + \tilde{p}_1 > 1$ return($\perp$)
5. With probability $\tilde{p}_0$ return $(0 \circ \text{UniGen}_{R'}(x, \alpha \circ 0))$
   With probability $\tilde{p}_1$ return $(1 \circ \text{UniGen}_{R'}(x, \alpha \circ 1))$
   Otherwise return($\perp$)

We invoke the algorithm by $\text{UniGen}_{R'}(x, \lambda)$, i.e. $\text{UniGen}_R(x) = \text{UniGen}_{R'}(x, \lambda)$.

6.3.5 Analysis

Let's begin with a definition

Definition 6.1 *We say that a call to* $\text{Approx}_{R'}(x,\alpha)$ *is* successful *if*

$$\text{Approx}_{R'}(x,\alpha) \in (1 \pm \epsilon)\,|R'(x,\alpha)|$$

First of all notice that

$$\Pr[\text{all calls to Approx are successful during the running of UniGen}] \geq 1 - \frac{1}{n}$$

since $\Pr[\text{Approx unsuccessful}] < \delta = \frac{1}{3n^2}$ and at each stage of the recursion we invoke Approx (via ShApprox) at most 3 times.
Now we'll begin establishing the main results.

Claim 6.3.1 *For all* $0 \leq i < n$ *if the call to* $\text{Approx}_{R'}(x,\alpha)$ *in* $\text{ShApprox}_{R'}(x,\alpha)$ *is successful then*

$$\left(1 + \frac{n-i-1}{n^2}\right) \cdot |R'(x,\alpha)| \leq \text{ShApprox}_{R'}(x,\alpha) \leq \left(1 + \frac{n-i}{n^2}\right) \cdot |R'(x,\alpha)|$$

Proof: For $0 \leq i < n$ we have

$$\text{ShApprox}_{R'}(x,\alpha) = \frac{4n^2+4n-4i}{4n^2+1} \cdot \text{Approx}_{R'}(x,\alpha)$$

so if $\text{Approx}_{R'}(x,\alpha) \in (1 \pm \epsilon)\,|R'(x,\alpha)|$ (for $\epsilon = \frac{1}{4n^2}$) then

$$\frac{4n^2+4n-4i}{4n^2+1} \cdot \left(1 - \frac{1}{4n^2}\right) \cdot |R'(x,\alpha)| \leq \text{ShApprox}_{R'}(x,\alpha) \leq \frac{4n^2+4n-4i}{4n^2+1} \cdot \left(1 + \frac{1}{4n^2}\right) \cdot |R'(x,\alpha)|$$

Now

$$\frac{4n^2+4n-4i}{4n^2+1} \cdot \left(1 + \frac{1}{4n^2}\right) = \frac{(4n^2+4n-4i)\cdot(4n^2+1)}{(4n^2+1)\cdot 4n^2} = 1 + \frac{n-i}{n^2}$$

and thus the upper bound follows.
For the lower bound we begin by noticing that

$$\frac{4n^2+4n-4i}{4n^2+1} \cdot \left(1 - \frac{1}{4n^2}\right) = \frac{(n^2+n-i)\cdot(4n^2-1)}{(4n^2+1)\cdot n^2}$$

Now since $(n^2+n-i)(4n^2-1) > (n^2+n-i-1)(4n^2+1)$, we get that

$$\frac{(n^2+n-i)(4n^2-1)}{(4n^2+1)n^2} > \frac{n^2+n-i-1}{n^2} = 1 + \frac{n-i-1}{n^2}$$

and the lower bound follows. ∎

Corollary 6.3.2 *If all calls to* Approx *during the running of* UniGen *are successful then in any stage of the recursion* $\tilde{p}_0 + \tilde{p}_1 \leq 1$

Proof: Assume all calls to Approx are successful. Let $i \stackrel{\text{def}}{=} |\alpha|$. Now if $0 \le i \le n-2$ then by the lemma above

$$\begin{aligned}
\text{ShApprox}_{R'}(x, \alpha \circ 0) \quad &+ \quad \text{ShApprox}_{R'}(x, \alpha \circ 1) \\
&\le \left(1 + \frac{n-i-1}{n^2}\right) \cdot |R'(x, \alpha \circ 0)| + \left(1 + \frac{n-i-1}{n^2}\right) \cdot |R'(x, \alpha \circ 1)| \\
&= \left(1 + \frac{n-i-1}{n^2}\right) \cdot |R'(x, \alpha)| \\
&\le \text{ShApprox}_{R'}(x, \alpha)
\end{aligned}$$

On the other hand, for $i = n-1$ we have for every $\sigma \in \{0,1\}$

$$|R'(x, \alpha \circ \sigma)| = \text{Round}(\text{Approx}_{R'}(x, \alpha \circ \sigma)) = \text{ShApprox}_{R'}(x, \alpha \circ \sigma)$$

where the first equality is due to the fact that $|R'(x, \alpha \circ \sigma)| \le 1$ and so rounding the value of the good approximation must yield exactly the correct value, which is preserved by ShApprox. Again, by Claim 6.3.1: $|R'(x, \alpha)| \le \text{ShApprox}_{R'}(x, \alpha)$, and so it follows that

$$\text{ShApprox}_{R'}(x, \alpha \circ 0) + \text{ShApprox}_{R'}(x, \alpha \circ 1) = |R'(x, \alpha)| \le \text{ShApprox}_{R'}(x, \alpha)$$

We conclude that in both cases

$$\tilde{p}_0 + \tilde{p}_1 = \frac{\text{ShApprox}_{R'}(x, \alpha \circ 0)}{\text{ShApprox}_{R'}(x, \alpha)} + \frac{\text{ShApprox}_{R'}(x, \alpha \circ 1)}{\text{ShApprox}_{R'}(x, \alpha)} \le 1$$

■

Lemma 6.3.3 *Assume that all calls to* Approx *are successful then for all x, α, β*

$$\Pr[\text{UniGen}_{R'}(x, \alpha) = \beta] = \begin{cases} \frac{1}{\text{ShApprox}_{R'}(x,\alpha)} & \text{if } \alpha \circ \beta \in R(x) \\ 0 & \text{otherwise} \end{cases}$$

furthermore if $\alpha \circ \beta \notin R(x)$ then $\Pr[\text{UniGen}_{R'}(x, \alpha) = \beta] = 0$ *regardless of the success of the* Approx *calls.*

Proof: First of all, we show that if $\alpha \circ \beta \notin R(x)$ then

$$\Pr[\text{UniGen}_{R'}(x, \alpha) = \beta] = 0$$

Observe that if $\alpha \circ \beta \notin R(x)$ then $R'(x, \alpha \circ \beta) = 0$. Now let $\beta = \gamma \circ \sigma$, where $|\gamma| = |\beta| - 1$ and look at the last recursive step of $\text{UniGen}_{R'}(x, \alpha)$ (i.e. the call to $\text{UniGen}_{R'}(x, \alpha \circ \gamma)$). If we have not arrived to this stage then the output was $\perp$, which is certainly not β. In case we did arrive at this step, the probability to output σ and thus finish outputting β as is

$$\tilde{p}_\sigma = \frac{\text{ShApprox}_{R'}(x, \alpha \circ \gamma \circ \sigma)}{\text{ShApprox}_{R'}(x, \alpha \circ \gamma)}$$

Now

$$\text{ShApprox}_{R'}(x, \alpha \circ \gamma \circ \sigma) = \text{Approx}_{R'}(x, \alpha \circ \gamma \circ \sigma) = 0$$

because by our definition of Approx we have that $\text{Approx}_{R'}(x, \alpha) = 0$ if $R'(x, \alpha) = 0$. Thus we get $\tilde{p}_\sigma = 0$ and so σ will not be output and the claim follows. Also notice that we have not used in any part of the above proof the assumption that all calls to Approx were successful.

We now turn to the more interesting case; that is, when $\alpha \circ \beta \in R(x)$. We will be proven the claim for this case by downward induction on $|\alpha|$. The claim is trivially true for $|\alpha| = n$ as in that case $\beta = \lambda$ and

$$\Pr\left[\mathrm{UniGen}_{R'}(x,\alpha) = \lambda\right] = 1 = \frac{1}{\mathrm{ShApprox}_{R'}(x,\alpha)}$$

Now we'll assume that the claim holds for $|\alpha| > k$ and prove it for $|\alpha| = k$. Let $\beta = \sigma \circ \beta'$ then

$$\Pr\left[\mathrm{UniGen}_{R'}(x,\alpha) = \beta\right] = \tilde{p}_\sigma \cdot \Pr\left[\mathrm{UniGen}_{R'}(x,\alpha\circ\sigma) = \beta'\right]$$

Now by the induction hypothesis

$$\Pr\left[\mathrm{UniGen}_{R'}(x,\alpha\circ\sigma) = \beta'\right] = \frac{1}{\mathrm{ShApprox}_{R'}(x,\alpha\circ\sigma)}$$

So we get that

$$\begin{aligned}
\Pr\left[\mathrm{UniGen}_{R'}(x,\alpha) = \beta\right] &= \tilde{p}_\sigma \cdot \frac{1}{\mathrm{ShApprox}_{R'}(x,\alpha\circ\sigma)} \\
&= \frac{\mathrm{ShApprox}_{R'}(x,\alpha\circ\sigma)}{\mathrm{ShApprox}_{R'_\alpha}(x)} \cdot \frac{1}{\mathrm{ShApprox}_{R'}(x,\alpha\circ\sigma)} \\
&= \frac{1}{\mathrm{ShApprox}_{R'}(x,\alpha)}
\end{aligned}$$

■

Corollary 6.3.4 *Assuming all calls to* Approx *are successful we have for all* $\beta \in R(x)$

$$\Pr\left[\mathrm{UniGen}_{R}(x) = \beta\right] = \frac{1}{\mathrm{ShApprox}_{R'}(x,\lambda)}$$

Proof: Take $\alpha = \lambda$ in Lemma 6.3.3. ■

Corollary 6.3.5 *If all calls to* Approx *are successful then for any* $\beta \in R(x)$ *we have*

$$\Pr\left[\mathrm{UniGen}_R(x) = \beta \mid \mathrm{UniGen}_R(x) \neq \perp\right] = \frac{1}{|R(x)|}$$

Proof: By Corollary 6.3.4 we have

$$\Pr\left[\mathrm{UniGen}_{R}(x) = \beta\right] = \frac{1}{\mathrm{ShApprox}_{R'}(x,\lambda)}$$

and this quantity is independent of β so all valid witnesses will be generated with the same probability. ■

Now we'll try to bound the probability that $\mathrm{UniGen}_{R'}(x,\alpha) = \perp$.

Lemma 6.3.6 *Under the assumption that all calls to* Approx *are successful we have that*

$$\Pr\left[\mathrm{UniGen}_{R}(x) = \perp\right] \leq \frac{2}{n}$$

Proof: By Claim 6.3.1 we know that

$$\left(1+\frac{n-i-1}{n^2}\right)\cdot|R'(x,\alpha)| \leq \mathrm{ShApprox}_{R'}(x,\alpha) \leq \left(1+\frac{n-i}{n^2}\right)\cdot|R'(x,\alpha)|$$

Thus we have that in each recursive step of the algorithm the following inequality holds:

$$\begin{aligned}
\tilde{p}_0+\tilde{p}_1 &= \frac{\mathrm{ShApprox}_{R'}(x,\alpha\circ 0)}{\mathrm{ShApprox}_{R'}(x,\alpha)}+\frac{\mathrm{ShApprox}_{R'}(x,\alpha\circ 1)}{\mathrm{ShApprox}_{R'}(x,\alpha)} \\
&\geq \frac{\left(1+\frac{n-i-2}{n^2}\right)\cdot(|R'(x,\alpha\circ 0)|+|R'(x,\alpha\circ 1)|)}{\left(1+\frac{n-i}{n^2}\right)\cdot|R'(x,\alpha)|} \\
&= \frac{n^2+n-i-2}{n^2+n-i} \geq 1-\frac{2}{n^2}
\end{aligned}$$

Since there are n recursion levels and by the above the probability of outputting $\perp$ at Stage 5 of the UniGen algorithm is smaller then $\frac{2}{n^2}$ at each recursion level, we conclude that

$$\Pr[\mathrm{UniGen}_R(x)=\perp] \leq n\cdot\frac{2}{n^2}=\frac{2}{n}$$

■

Corollary 6.3.7

$$\Pr[\mathrm{UniGen}_R(x)=\perp] \leq \frac{3}{n}$$

Proof: Let AS denote the event *All calls to* Approx *are Successful.*
Using Lemma 6.3.6 and the fact that $\Pr[\mathrm{AS}] \geq 1-\frac{1}{n}$ we have that

$$\begin{aligned}
\Pr[\mathrm{UniGen}_R(x)=\perp] &= \Pr[(\mathrm{UniGen}_R(x)=\perp)\wedge\mathrm{AS}]+\Pr\left[(\mathrm{UniGen}_R(x)=\perp)\wedge\overline{\mathrm{AS}}\right] \\
&\leq \Pr[\mathrm{UniGen}_R(x)=\perp\mid\mathrm{AS}]+\Pr\left[\overline{\mathrm{AS}}\right] \\
&\leq \frac{2}{n}+\frac{1}{n}
\end{aligned}$$

■

Combining Corollaries 6.3.5 and 6.3.7, we conclude that the procedure UniGen_R behaves as desired.

6.4 An alternate algorithm for uniform generation

6.4.1 Intuition

Now we shall sketch an alternative algorithm for uniform generation. In addition to using approximate counting this algorithm will use hash functions in a direct way. The hash functions will be of high independence (and thus have stronger "random properties") than pairwise independence used before.

6.4.2 Definitions

Let $H(n,m,t)$ denote a collection of t-wise independent hash functions of n bits to m bits. This means that for any $y_1,\dots,y_t \in \{0,1\}^m$ and any distinct $x_1,\dots,x_t \in \{0,1\}^n$ we have for a hash function h uniformly chosen from $H(n,m,t)$ that

$$\Pr\left[h(x_1)=y_1 \wedge \ldots \wedge h(x_t)=y_t\right] = 2^{-mt}$$

Let $h \in H(n,m,t)$ then for $\alpha \in \{0,1\}^m$ we let $h^{-1}(\alpha) = \{y \in \{0,1\}^n \mid h(y)=\alpha\}$.
Recall that $S = R(x)$ and $n = |x|$.

6.4.3 Algorithm $\mathrm{UniGen}_R(x)$

1. If $|S| \le 2n^5$ compute a listing $y_1,\dots,y_s$ of the members of the set $|S|$, select j at random from $1,\dots,s$, output y_j and halt.
2. Otherwise, let $N \approx |S|$ (we obtain a good approximation of $|S|$ using approximate counting), and let $m = \log_2 N - 5\log n$
3. Select h uniformly in $H(n,m,n)$
4. If $\exists \alpha \in \{0,1\}^m$ s.t. $|S \cap h^{-1}(\alpha)| \ge 2n^5$ then output $\perp$
5. Select α uniformly in $\{0,1\}^m$
6. Compute $S \cap h^{-1}(\alpha)$ and for every $y \in S \cap h^{-1}(\alpha)$ output it with probability $\frac{1}{2n^5}$, otherwise output $\perp$

The implementation of Steps $1,4,6$ involves the use of an NP-oracle. In each case we build an appropriate NP-relations and use the oracle to check membership in those relations.
For example in Step 4 we would use the following NP-relation and corresponding NP-set:

$$S'' = \{(x,h) : \exists \alpha, y_1,\dots,y_k \text{ such that } ((x,h),(\alpha,y_1,\dots,y_k)) \in R''\}$$

$$R'' = \left\{ ((x,h),(\alpha,y_1,\dots,y_k)) : \begin{array}{l} k = 2|x|^5 \text{ and } y_1,\dots,y_k \text{ are distinct} \\ \text{and } h \in H(|x|,m,|x|) \\ \text{and } R(x,y_i)=1 \text{ (for every } i \in [k]) \\ \text{and } h(y_i)=\alpha \text{ (for every } i \in [k]) \end{array} \right\}$$

6.4.4 Analysis

To show that the algorithm indeed is a uniform generator we note that if $|S| \le 2n^5$ then the algorithm halts in Step 1 outputting a uniformly chosen element of S. If $|S| > 2n^5$ then considering any possible (fixed) choice of h in Step 3 and assuming a non-$\perp$ result in Step 4 we have that the randomization is only over the choice of α in step 5 and the choices in step 6. And so we have that the probability that any fixed $y \in S$ will be output is:

$$\begin{aligned} \Pr[\text{y is output}] &= \Pr\left[h(y)=\alpha : \alpha \xleftarrow{R} \{0,1\}^m\right] \cdot \frac{1}{2n^2} \\ &= 2^{-m} \cdot \frac{1}{2n^2} \end{aligned}$$

and this quantity is independent of y.

Another important claim is that the probability of a non-$\perp$ output is a positive constant. First notice that if h is drawn at random from $H(n,m,n)$ then the expected size of $|S \cap h^{-1}(\alpha)|$ is $\frac{|S|}{2^m} = n^5$, and from this fact it's possible to show via standard t-wise independence techniques that the probability that $|S \cap h^{-1}(\alpha)| < 2n^5$ for all α is a positive constant. We conclude that the probability of not halting in Step 4 of the algorithm is a positive constant.

Conditioned on not halting in Step 4, the probability that any fixed $y \in S$ is output in Step 6 is $2^{-m} \cdot \frac{1}{2n^5}$, so it follows that the probability of outputting $\perp$ in Step 6 equals

$$1 - |S| \cdot \frac{2^{-m}}{2n^5} = 1 - |S| \cdot \frac{2^{5\log_2 n - log_2 N}}{2n^5} = 1 - \frac{|S|}{2N}$$

Now since N is a good approximation of $|S|$, we have $\frac{|S|}{2N} > \frac{1}{3}$. Thus the probability of outputting an element of S in Step 6 is at least $1/3$. Finally, as we have shown that the probability of halting in Step 4 is a positive constant, and the probability of a non-$\perp$ result in Step 6 is a positive constant, we conclude that the probability of the algorithm to output an element of S is a positive constant.

Bibliographic Notes

The relation between approximate counting and uniform generation was first explicitly studied by Jerrum, Valiant and Vazirani [28]. The direct presentation of a uniform generation procedure is due to Bellare, Goldreich and Petrank [12].

Lecture 7

Small Bias Sample Spaces (Part 1)

Notes taken by Yehuda Lindell and Alon Rosen

Summary: In this lecture we introduce the notion of ϵ-bias sample spaces. Informally, a random variable over $\{0,1\}^n$ is ϵ-bias if for every subset of bits, the difference between the probability that the parity of the bits is zero and the probability it is one is at most ϵ. We show a connection between ϵ-bias and the statistical distance to the uniform distribution. Next, we present an application of ϵ-bias sample spaces for generating "approximately" k-wise independent distributions (i.e., distributions over $\{0,1\}^n$ in which every k bits look almost uniformly distributed over $\{0,1\}^k$).

7.1 Introduction

In previous lectures, the notion of k-wise independent distributions was considered. Such distributions are a relaxation of the uniform distribution in that any subset of size k is uniformly distributed (and yet nothing can be said of subsets of size larger than k). The motivation for considering such relaxations is the desire to obtain (efficiently constructible) "small" sample spaces (where by small, we mean of size polynomial in n where, for example, the space is distributed over $\{0,1\}^n$).

In this lecture, we introduce a different relaxation of uniform distributions, called ϵ-bias. Loosely speaking, we say that a random variable (over $\{0,1\}^n$) is ϵ-bias if for *every* subset of bits, the difference between the probability that the parity of the bits is 0 and 1 is at most ϵ. This is a natural relaxation of the uniform distribution, for which the probability that any subset of bits has parity 0 equals exactly 1/2 (likewise for parity 1). In other words, the uniform distribution has zero bias. As we shall see in this lecture, the converse is also true and thus a random variable is uniformly distributed *if and only if* it has zero bias. Therefore, the notion of bias can be used to fully characterize the uniform distribution.

As we have mentioned, a zero-bias sample space is uniform. A natural question to ask is how to characterize the distribution of an ϵ-bias sample space, for ϵ that is strictly greater than zero. Indeed, the bulk of this lecture focuses on understanding the "distance" of an ϵ-bias sample space from the uniform distribution (where the notion of distance used is the classical one of statistical difference).

We also present an application of ϵ-bias sample spaces (another more "direct" application is presented in the next lecture). Loosely speaking, we show that "almost" k-wise independent spaces can be derived from ϵ-bias sample spaces, where almost k-wise independent spaces are such that when considering any subset of size k, the resulting distribution is "approximately" uniform (rather than being truly uniform as in the case of k-wise independence). We note that such spaces can be made much smaller than is possible for (fully) k-wise independent spaces.

7.2 Definitions

7.2.1 Statistical Difference

We begin by defining a natural measure of the distance between distributions. This measure, called the statistical difference (or variation distance) is the traditional way of measuring the distance between distributions. We note that, as discussed in Lecture 1, we abuse notation and view a random variable as a distribution (or alternatively, a sample space), rather than as a function mapping n-bit strings to $[0,1]$.

Definition 7.1 (statistical difference): *Let $X, Y \in \{0,1\}^n$ be random variables. The* statistical difference *between X and Y is defined by the function:*

$$\Delta(X,Y) \stackrel{\text{def}}{=} \frac{1}{2} \cdot \sum_{\alpha \in \{0,1\}^n} |\Pr[X = \alpha] - \Pr[Y = \alpha]|$$

Observe that removing the absolute value from the definition would render it meaningless (as it would always sum to zero). A seemingly strange part of this definition is the constant factor of $1/2$. Firstly, this factor provides a normalization of the statistical difference in that for every X and Y, it holds that $0 \leq \Delta(X,Y) \leq 1$. Secondly, including this factor results in the above definition being equivalent to the following one:

Definition 7.2 (statistical difference – alternative definition): *Let $X, Y \in \{0,1\}^n$ be random variables. The* statistical difference *between X and Y is defined by,*

$$\Delta(X,Y) \stackrel{\text{def}}{=} \max_{S \subseteq \{0,1\}^n} \{\Pr[X \in S] - \Pr[Y \in S]\}$$

Proposition 7.2.1 *Definitions 7.1 and 7.2 are equivalent. That is, for all random variables X and Y, $\Delta_1(X,Y) = \Delta_2(X,Y)$, where Δ_i denotes the statistical difference by Definition i.*

Proof: We begin by observing that the set S for which the value $(\Pr[X \in S] - \Pr[Y \in S])$ is maximum has the following property:

1. For every $\alpha \in S$, it holds that $\Pr[X = \alpha] \geq \Pr[Y = \alpha]$, and
2. For every $\alpha \notin S$, it holds that $\Pr[X = \alpha] \leq \Pr[Y = \alpha]$.

Intuitively, the above property holds because otherwise an α contradicting it could be moved from S to $\overline{S}$ (or the reverse) and a "larger than maximum" value would be obtained. That is, if there exists an $\alpha \in S$ for which $\Pr[X = \alpha] < \Pr[Y = \alpha]$, then it is easy to verify that

$$\Pr[X \in S \setminus \{\alpha\}] - \Pr[Y \in S \setminus \{\alpha\}] > \Pr[X \in S] - \Pr[Y \in S]$$

in contradiction to the maximality of the value $(\Pr[X \in S] - \Pr[Y \in S])$ induced by S. An analogous argument holds for item (2) of the above property.

We now claim that for every set S for which the above property holds, we have that

$$\Pr[X \in S] - \Pr[Y \in S] = \frac{1}{2} \cdot \sum_{\alpha \in \{0,1\}^n} |\Pr[X = \alpha] - \Pr[Y = \alpha]| = \Delta_1(X, Y) \tag{7.1}$$

Eq. (7.1) follows from the following simple calculation:

$$\begin{aligned}
\Pr[X \in S] - \Pr[Y \in S] &= \frac{1}{2} \cdot (\Pr[X \in S] - \Pr[Y \in S] - \Pr[Y \in S] + \Pr[X \in S]) \\
&= \frac{1}{2} \cdot (\Pr[X \in S] - \Pr[Y \in S] + \Pr[Y \notin S] - \Pr[X \notin S]) \\
&= \frac{1}{2} \cdot \left(\sum_{\alpha \in S} (\Pr[X = \alpha] - \Pr[Y = \alpha]) + \sum_{\alpha \notin S} (\Pr[Y = \alpha] - \Pr[X = \alpha]) \right) \\
&= \frac{1}{2} \cdot \sum_{\alpha \in \{0,1\}^n} |\Pr[X = \alpha] - \Pr[Y = \alpha]|
\end{aligned}$$

where the last equality is due to the fact that the above property holds with respect to S. This concludes the proof of the proposition because we have that for S for which $\Pr[X \in S] - \Pr[Y \in S]$ is maximum, Eq. (7.1) holds. This implies that $\Delta_1(X, Y) = \Delta_2(X, Y)$. ■

Notation: We denote by U_n the uniform distribution over n-bit strings. In this lecture, we are interested in distributions X such that $\Delta(X, U_n)$ is small (clearly, $\Delta(X, U_n) = 0$ if and only if $X = U_n$).

An advanced comment: The sets S in Definition 7.2 can be viewed as (not necessarily efficiently computable) tests for the statistical difference between X and Y. That is, for each set S, associate a distinguisher D_S who upon input α outputs 1 if and only if $\alpha \in S$. Then, it follows that

$$\Pr[D_S(X) = 1] - \Pr[D_S(Y) = 1] = \Pr[X \in S] - \Pr[Y \in S]$$

However, notice that such a distinguisher may not be efficient (i.e., polynomial-time). This is because there may not necessarily be efficient procedures for recognizing whether or not $\alpha \in S$.

The notion of computational indistinguishability, in a specific incarnation that states that no polynomial-size circuit can distinguish between two distribution ensembles $\{X_n\}_{n \in \mathsf{N}}$ and $\{Y_n\}_{n \in \mathsf{N}}$ (where for every n, X_n and Y_n range over strings of length polynomial in n), is a relaxed notion of statistical closeness. (We say that $\{X_n\}$ and $\{Y_n\}$ are statistically close if $\Delta(X_n, Y_n)$ is a negligible function in n.) To see this, first notice that for every circuit C_n and every n, it holds that C_n can distinguish X_n from Y_n with probability at most $\Delta(X_n, Y_n)$. This is because we can define $S_n = \{\alpha \mid C_n(\alpha) = 1\}$ and thus it follows that for every n,

$$|\Pr[C_n(X_n) = 1] - \Pr[C_n(Y_n) = 1]| = |\Pr[X_n \in S_n] - \Pr[Y_n \in S_n]| \leq \Delta(X_n, Y_n)$$

From this, we can conclude that if $\{X_n\}$ and $\{Y_n\}$ are statistically close, then they are also computationally indistinguishable. (Notice that the notion of computational indistinguishability, as described above, refers to polynomial-size circuits. Nevertheless, this implies computational indistinguishability by probabilistic polynomial-time algorithms, since one may hardwire into a circuit the algorithm's "best" coin-tosses. Thus, we are guaranteed that the distinguishing gap of the circuit is at least as good as that of the algorithm.)

7.2.2 ϵ-Bias Sample Spaces

We introduce the notion of ϵ-bias sample spaces or random variables. (We will move freely between the notions of sample spaces and random variables, with the intention being that the random variable is defined according to the distribution of the corresponding sample space.) Loosely speaking, the bias of a random variable, with respect to a subset of indices I, equals the difference between the probability that the parity of the bits indexed by I equals zero and the probability that it equals one. An ϵ-bias random variable is one for which the bias (over all subsets of indices) is at most ϵ. In some sense this has the flavor of uniformity in that the parity of any subset of bits of a uniformly distributed string is unbiased to 0 or 1. In fact, the following two conditions are equivalent:

1. The bits of the random variable $X = X_1, \ldots, X_n$ are independent and uniformly distributed.
2. For every subset of bits of X, it is equally likely that the parity of the subset is either zero or one.

(The fact that the above two properties are equivalent is proven in the next section, see Corollary 7.6.) Therefore, ϵ-bias can be viewed as some measure of distance from the uniform distribution. We stress that although there is some connection between ϵ-bias and statistical distance from uniform, the notions are not at all equivalent.

Definition 7.3 (bias): *Let $X = X_1, \ldots, X_n \in \{0,1\}^n$ be a random variable, and let $I \subseteq [n]$ (where $[n]$ denotes the set $\{1, \ldots, n\}$). The* bias *of the random variable X, relative to the subset I, is defined by,*

$$\text{bias}_I(X) \stackrel{\text{def}}{=} \Pr\left[\bigoplus_{i\in I} X_i = 0\right] - \Pr\left[\bigoplus_{i\in I} X_i = 1\right]$$

where by convention for any $(x_1, \ldots, x_n)$, it holds that $\bigoplus_{i\in\emptyset} x_i \equiv 0$.

An equivalent definition of bias is given by:

$$\text{bias}_I(X) \stackrel{\text{def}}{=} 2 \cdot \Pr\left[\bigoplus_{i\in I} X_i = 0\right] - 1 \tag{7.2}$$

The equivalence is derived from the trivial fact that $\Pr[\bigoplus_{i\in I} X_i = 1] = 1 - \Pr[\bigoplus_{i\in I} X_i = 0]$. Notice that $-1 \le \text{bias}_I(X) \le 1$ always holds.

The above definition of bias refers to a *specific* subset of bits of X. We now present the definition of max-bias, which extends this to all subsets of X.

Definition 7.4 (max-bias and ϵ-bias sample spaces): *Let $X \in \{0,1\}^n$ be a random variable. The* max-bias *of X is defined by,*

$$\text{maxbias}(X) \stackrel{\text{def}}{=} \max_{\substack{I\subseteq[n]\\ I\neq\emptyset}} \{|\text{bias}_I(X)|\}$$

*We say that X is an ϵ-*bias sample space *if* $\text{maxbias}(X) \le \epsilon$.

Clearly, the max-bias of a uniform space equals 0 (i.e., $\text{maxbias}(U_n) = 0$). As we shall see, the converse is also true. However, as we have mentioned above, it is not true that an ϵ-bias sample space X satisfies $\Delta(X, U_n) \leq \epsilon$. Despite this, there is a connection between these notions (as stated in Theorem 7.5 below).

7.3 Statistical Difference versus Maxbias

The following theorem demonstrates the connection between ϵ-bias and statistical difference. In particular, it implies that if X is ϵ-bias, then X is at most $\frac{\epsilon \cdot \sqrt{2^n}}{2}$-far from uniform.

Theorem 7.5 *Let $X \in \{0,1\}^n$ be a random variable. Then,*

$$\text{maxbias}(X) \leq 2 \cdot \Delta(X, U_n) \leq \sqrt{2^n} \cdot \text{maxbias}(X)$$

We remark that an immediate corollary of Theorem 7.5 is that a random variable X is uniformly distributed (i.e., $\Delta(X, U_n) = 0$) if and only if $\text{maxbias}(X) = 0$. Thus, the notion of bias offers an alternative characterization of the uniform distribution.

Corollary 7.6 *A random variable X is uniform if and only if* $\text{maxbias}(X) = 0$.

7.3.1 Proof of Theorem 7.5 (Main Result)

We first show that

$$\text{maxbias}(X) \leq 2 \cdot \Delta(X, U_n) \tag{7.3}$$

Let $\epsilon = \text{maxbias}(X)$. Then, there exists a non-empty subset $I \subseteq [n]$ such that $\text{bias}_I(X) = \epsilon$. Now, define $S_I = \{\alpha \mid \bigoplus_{i \in I} \alpha_i = 0\}$. Intuitively, this subset of strings (for which X is biased towards zero) constitutes a "test" for the statistical difference by Definition 7.2. Specifically,

$$\Pr\left[X \in S_I\right] = \Pr\left[\bigoplus_{i \in I} \alpha_i = 0\right] = \frac{1}{2} + \frac{\epsilon}{2}$$

where the second equality is from the alternative definition of ϵ-bias as shown in Eq. (7.2). On the other hand, clearly $\Pr[U_n \in S_I] = \frac{1}{2}$. Therefore, we have found a set S for which

$$\Pr[X \in S] - \Pr[U_n \in S] = \frac{\epsilon}{2}$$

Eq. (7.3) now follows. (We note that it is not necessarily true that $\text{maxbias}(X) = 2 \cdot \Delta(X, U_n)$ because there may be other sets S, for which the difference $\Pr[X \in S] - \Pr[U_n \in S]$ is even greater than $\epsilon/2$.)

We now show that

$$2 \cdot \Delta(X, U_n) \leq \sqrt{2^n} \cdot \text{maxbias}(X) \tag{7.4}$$

In general, probability distributions over $\{0,1\}^n$ are viewed as functions f mapping n-bit strings to $[0,1]$ (and satisfy $\sum_\alpha f(\alpha) = 1$). An alternative view that is instructive for this proof, is to represent such functions as vectors in $\mathbb{R}^N$, where $N = 2^n$ (and $\mathbb{R}$ denotes the set of real numbers). According to this view, the length-N vector representation of a function f is such that the coordinate indexed by $x \in \{0,1\}^n$ contains the value $f(x)$. Specifically, the set $f : \{0,1\}^n \to \mathbb{R}$ of real functions on $\{0,1\}^n$ forms an N-dimensional real vector space where vectors in this space are of the form:

$$v_f \stackrel{\text{def}}{=} \langle f(x) \rangle_{x \in \{0,1\}^n}$$

We call such a vector a probability vector if all its entries are non-negative and they sum-up to 1. From now on, we associate functions f with vectors v_f of the above form. The inner-product of two vectors (functions) f and g is defined by:

$$\langle f, g \rangle = \sum_{x \in \{0,1\}^n} f(x) \cdot g(x)$$

The above way of viewing probability distributions turns out to be useful when analyzing sample spaces. As we will see, there is a correspondence between different bases of such vector spaces and the notions of max-bias and statistical difference. Then, using properties of linear algebra, we derive Eq. (7.4). We consider two different N-dimensional bases for the above vector space:

1. **Kronecker Basis:** The Kronecker Basis, denoted $\mathcal{K} = \{k_\alpha\}_{\alpha \in \{0,1\}^n}$ is defined as follows:

   $$k_\alpha(x) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } x = \alpha \\ 0 & \text{if } x \neq \alpha \end{cases}$$

   This is the standard basis for vector spaces (in vector notation this is the basis in which the i^{th} element of the basis is the vector with a single 1 in the i^{th} place and zeros elsewhere).

2. **Fourier Basis:** The Fourier Basis, denoted $\mathcal{F} = \{f_I\}_{I \subseteq [n]}$ is defined as follows:

   $$f_I(x) \stackrel{\text{def}}{=} \frac{1}{\sqrt{N}} \cdot (-1)^{\sum_{i \in I} x_i}$$

In the proof we switch freely between viewing a random variable X as a distribution on $\{0,1\}^n$ (that is, a sample space) and viewing it as a probability function $\pi : \{0,1\}^n \to [0,1]$ (representing the probability that a specific string in $\{0,1\}^n$ is chosen in the sample space).

We now show that both the Kronecker and Fourier bases are orthonormal (this technicality is needed later in the proof).

Fact 7.3.1 *The Kronecker basis is* orthonormal. *That is, for every $\alpha \neq \beta \in \{0,1\}^n$:*

$$\langle k_\alpha, k_\alpha \rangle = 1 \quad \text{and} \quad \langle k_\alpha, k_\beta \rangle = 0$$

Proof: Note that for every $\alpha, \beta \in \{0,1\}^n$:

$$\begin{aligned} \langle k_\alpha, k_\beta \rangle &= \sum_{x \in \{0,1\}^n} k_\alpha(x) \cdot k_\beta(x) \\ &= k_\alpha(\beta) \end{aligned}$$

where the last equality is due to the fact that $k_\beta(\beta) = 1$, whereas for every $x \neq \beta$, $k_\beta(x) = 0$. This completes the proof, because $k_\alpha(\alpha) = 1$ and $k_\alpha(\beta) = 0$ for every $\beta \neq \alpha$. ■

Fact 7.3.2 *The Fourier basis is* orthonormal. *That is, for every $I \neq J \subseteq [n]$:*

$$\langle f_I, f_I \rangle = 1 \quad \text{and} \quad \langle f_I, f_J \rangle = 0$$

Proof: Let $I \subseteq [n]$. Then:

$$\begin{aligned}
\langle f_I, f_I \rangle &= \sum_{x\in\{0,1\}^n} f_I(x)\cdot f_I(x) \\
&= \tfrac{1}{N}\cdot \sum_{x\in\{0,1\}^n} (-1)^{\sum_{i\in I} x_i}\cdot(-1)^{\sum_{i\in I} x_j} \\
&= \tfrac{1}{N}\cdot \sum_{x\in\{0,1\}^n} (-1)^{2\sum_{i\in I} x_i} \\
&= \tfrac{1}{N}\cdot \sum_{x\in\{0,1\}^n} 1 \\
&= \tfrac{1}{N}\cdot N
\end{aligned}$$

as required. On the other hand, for $I \neq J \subseteq [n]$, we have:

$$\begin{aligned}
\langle f_I, f_J \rangle &= \sum_{x\in\{0,1\}^n} f_I(x)\cdot f_J(x) \\
&= \tfrac{1}{N}\cdot \sum_{x\in\{0,1\}^n} (-1)^{\sum_{i\in I} x_i}\cdot(-1)^{\sum_{j\in J} x_j} \\
&= \tfrac{1}{N}\cdot \sum_{x\in\{0,1\}^n} (-1)^{\sum_{i\in I\nabla J} x_i} \qquad (7.5)
\end{aligned}$$

where $I\nabla J$ denotes the symmetric difference between sets I and J (that is, $I\nabla J \stackrel{\text{def}}{=} (I\setminus J)\cup(J\setminus I)$). Notice that $I\nabla J \neq \emptyset$ (since $I \neq J$), and that the sum in Eq. (7.5) ranges over all $x \in \{0,1\}^n$. For, $b \in \{0,1\}$, consider the sets $S^b_{I\nabla J} = \{x \mid \oplus_{i\in I\nabla J}\, x_i = b\}$. Using this notation, we have:

$$\begin{aligned}
\tfrac{1}{N}\cdot \sum_{x\in\{0,1\}^n} (-1)^{\sum_{i\in I\nabla J} x_i} &= \tfrac{1}{N}\cdot\left(\sum_{x\in S^0_{I\nabla J}} (-1)^{\sum_{i\in I\nabla J} x_i} + \sum_{x\in S^1_{I\nabla J}} (-1)^{\sum_{i\in I\nabla J} x_i}\right) \\
&= \tfrac{1}{N}\cdot\left(\left|S^0_{I\nabla J}\right|\cdot 1 - \left|S^1_{I\nabla J}\right|\cdot 1\right)
\end{aligned}$$

The key observation is that if $I\nabla J \neq \emptyset$, then the size of $S^0_{I\nabla J}$ equals the size of $S^1_{I\nabla J}$. This can be seen by showing a bijection between the two sets. A simple such bijection is a function that flips the i^{th} bit of $x \in S^0_{I\nabla J}$, where $i \in I\nabla J$. That is, for an arbitrary $i \in I\nabla J$ define,

$$F_i(x_1\cdots x_n) \stackrel{\text{def}}{=} x_1\cdots x_{i-1}\cdot\overline{x}_i\cdot x_{i+1}\cdots x_n$$

Clearly, for $x \in S^0_{I\nabla J}$, we have $F_i(x) \in S^1_{I\nabla J}$ (the fact that F_i is a bijection is trivial). We conclude that,

$$\begin{aligned}
\langle f_I, f_J \rangle &= \tfrac{1}{N}\cdot\left(\left|S^0_{I\nabla J}\right| - \left|S^1_{I\nabla J}\right|\right) \\
&= \tfrac{1}{N}\cdot 0
\end{aligned}$$

as required. ■

In the following fact, we show that the bias of a probability function π, relative to the subset I, can be stated in terms of the inner-product of π with f_I.

Fact 7.3.3 *For every $I \subseteq [n]$ and for every probability function $\pi : \{0,1\}^n \to [0,1]$,*

$$\mathrm{bias}_I(\pi) = \sqrt{N} \cdot \langle \pi, f_I \rangle$$

Proof: The key observation for proving the fact is that for $b \in \{0,1\}$, the probability that the parity of x equals b can be computed by summing $\pi(x)$ for all strings x for which the parity of x is indeed b. Denote the set of strings x, for which the parity of the bits of x indexed by I equals b, by S_I^b. That is, $S_I^b \stackrel{\mathrm{def}}{=} \{x \mid \bigoplus_{i \in I} x_i = b\}$. Then, for $b \in \{0,1\}$ and random variable $X = (X_1, \ldots, X_n)$ with distribution π:

$$\Pr\left[\bigoplus_{i \in I} X_i = b\right] = \sum_{x \in S_I^b} \pi(x)$$

We thus have,

$$\begin{aligned} \mathrm{bias}_I(\pi) &= \left(\sum_{x \in S_I^0} \pi(x)\right) - \left(\sum_{x \in S_I^1} \pi(x)\right) \\ &= \sum_{x \in \{0,1\}^n} \pi(x) \cdot (-1)^{\sum_{i \in I} x_i} \end{aligned}$$

where the second equality is due to the fact that for $x \in S_I^b$, it holds that $(-1)^{\sum_{i \in I} x_i} = (-1)^b$, and thus $\pi(x)(-1)^{\sum_{i \in I} x_i} = \pi(x)(-1)^b$. Now, by the definition of the Fourier basis (and the inner-product), it follows that,

$$\begin{aligned} \mathrm{bias}_I(\pi) &= \sum_{x \in \{0,1\}^n} \pi(x) \cdot \sqrt{N} \cdot f_I(x) \\ &= \sqrt{N} \cdot \langle \pi, f_I \rangle \end{aligned}$$

as required. ■

Given Fact 7.3.3, we are ready to show a connection between $\mathrm{maxbias}(\pi)$ and the $L_\infty^{\mathcal{F}}$ norm (i.e., the L_∞ norm with respect to the Fourier basis $\mathcal{F}$). Following this, we show a connection between $\Delta(\pi, U_n)$ and the $L_1^{\mathcal{K}}$ norm (i.e., the L_1 norm with respect to the Kronecker basis $\mathcal{K}$). We then conclude by presenting a fact from linear algebra connecting between the L_1 and L_∞ norms. We first recall the definition of $L_1^{\mathcal{B}}$ and $L_\infty^{\mathcal{B}}$, for a basis $\mathcal{B}$:

$$\begin{aligned} \| v \|_1^{\mathcal{B}} &\stackrel{\mathrm{def}}{=} \sum_{b \in \mathcal{B}} |\langle v, b \rangle| \\ \| v \|_\infty^{\mathcal{B}} &\stackrel{\mathrm{def}}{=} \max_{b \in \mathcal{B}} \{|\langle v, b \rangle|\} \end{aligned}$$

In general for any integer p (e.g., $p = 2$) the $L_p^{\mathcal{B}}$ norm is defined as:

$$\| v \|_p^{\mathcal{B}} \stackrel{\mathrm{def}}{=} \left(\sum_{b \in \mathcal{B}} |\langle v, b \rangle|^p\right)^{1/p}$$

Now, define the constant function $\mu : \{0,1\}^n \to \mathbb{R}$ such that for every $x \in \{0,1\}^n$, $\mu(x) = \frac{1}{N}$. The function μ is the probability function associated with the uniform distribution U_n. The following claim establishes the connection between the maxbias of a probability function π and the $L_\infty^{\mathcal{F}}$ norm of π (or, actually of $\pi - \mu$).

Claim 7.3.4 *For every probability function* $\pi : \{0,1\}^n \to \mathbb{R}$*,*

$$\mathrm{maxbias}(\pi) = \sqrt{N}\cdot \parallel \pi - \mu \parallel_\infty^{\mathcal{F}}$$

Proof: By Fact 7.3.3 we have that

$$\mathrm{maxbias}(\pi) \stackrel{\mathrm{def}}{=} \max_{\substack{I\subseteq[n]\\ I\neq\emptyset}} \{|\mathrm{bias}_I(\pi)|\} = \sqrt{N}\cdot \max_{\substack{I\subseteq[n]\\ I\neq\emptyset}} \{|\langle \pi, f_I\rangle|\} \tag{7.6}$$

We now make the following two observations regarding the Fourier basis:

1. For every non-empty subset $I \subseteq [n]$, $\langle \mu, f_I\rangle = 0$.
   This is due to the fact that $\langle \mu, f_I\rangle = \frac{1}{N\sqrt{N}} \sum_x (-1)^{\sum_{i\in I} x_i}$. As we have previously seen (in the proof of Fact 7.3.2), for any non-empty subset I, the above sum equals zero.
2. For every probability vector v, $\langle v, f_\emptyset\rangle = \frac{1}{\sqrt{N}}$ (by the definition of the Fourier basis and the fact that the sum of the elements of v equals 1).

This implies that for every $I \subseteq [n]$,

$$\langle \pi - \mu, f_I\rangle = \langle \pi, f_I\rangle - \langle \mu, f_I\rangle = \begin{cases} \langle \pi, f_I\rangle & \text{if } I \neq \emptyset \\ 0 & \text{if } I = \emptyset \end{cases} \tag{7.7}$$

Then, combining Eq. (7.6) and(7.7) we obtain that

$$\mathrm{maxbias}(\pi) \;=\; \sqrt{N}\cdot \max_{\substack{I\subseteq[n]\\ I\neq\emptyset}} \{|\langle \pi - \mu, f_I\rangle|\} \;=\; \sqrt{N}\cdot \parallel \pi - \mu \parallel_\infty^{\mathcal{F}}$$

as required. ■

We now turn to establish the connection between the statistical difference of π from U_n and the $L_1^{\mathcal{K}}$ norm of $\pi - \mu$.

Claim 7.3.5 *For every probability function* $\pi : \{0,1\}^n \to \mathbb{R}$*:*

$$2\cdot \Delta(\pi, U_n) = \parallel \pi - \mu \parallel_1^{\mathcal{K}}$$

Proof: Recall that for every vector v, $\langle v, k_\alpha\rangle = v(\alpha)$. Thus:

$$\begin{aligned} 2\cdot\Delta(\pi, U_n) &= \sum_{x\in\{0,1\}^n} |\pi(x) - \mu(x)| \\ &= \sum_{x\in\{0,1\}^n} |\langle \pi, k_x\rangle - \langle \mu, k_x\rangle| \\ &= \sum_{x\in\{0,1\}^n} |\langle \pi - \mu, k_x\rangle| \\ &= \parallel \pi - \mu \parallel_1^{\mathcal{K}} \end{aligned}$$

as required. ■

We conclude the proof by combining Claims 7.3.4 and 7.3.5 with the following technical lemma from linear algebra (proven below).

Lemma 7.3.6 *For every two orthonormal bases $\mathcal{A}$ and $\mathcal{B}$, and for every vector v:*

$$\| v \|_1^{\mathcal{A}} \leq N \cdot \| v \|_\infty^{\mathcal{B}}$$

Using Lemma 7.3.6 we have:

$$\begin{aligned} 2 \cdot \Delta(\pi, U_n) &= \| \pi - \mu \|_1^{\mathcal{K}} \\ &\leq N \cdot \| \pi - \mu \|_\infty^{\mathcal{F}} \\ &= \sqrt{N} \cdot \mathrm{maxbias}(\pi) \end{aligned}$$

Eq. (7.4) follows. Thus, all that remains to complete the proof of Theorem 7.5 is to prove Lemma 7.3.6.

7.3.2 Proof of Lemma 7.3.6 (Technical Lemma from Linear Algebra)

In order to prove Lemma 7.3.6 we will make use of the following three facts:

Fact 7.3.7 *For every basis $\mathcal{A}$:*

$$\| v \|_1^{\mathcal{A}} \leq \sqrt{N} \cdot \| v \|_2^{\mathcal{A}}$$

Proof: We start by recalling the Cauchy-Schwartz inequality. Let $\{\alpha_i\}_{i=1}^N$ and $\{\beta_i\}_{i=1}^N$ be two sequences of real numbers. Then:

$$\left(\sum_{i=1}^N \alpha_i \cdot \beta_i \right)^2 \leq \left(\sum_{i=1}^N \alpha_i^2 \right) \cdot \left(\sum_{i=1}^N \beta_i^2 \right)$$

Now, let $\mathcal{A} = \{a_i\}_{i=1}^N$ be a basis. Then, by the definition of the $L_1^{\mathcal{A}}$ and $L_2^{\mathcal{A}}$ norms and by the Cauchy-Schwartz inequality:

$$\begin{aligned} \left(\| v \|_1^{\mathcal{A}} \right)^2 &= \left(\sum_{i=1}^N |\langle v, a_i \rangle| \right)^2 \\ &\leq \left(\sum_{i=1}^N 1^2 \right) \cdot \left(\sum_{i=1}^N |\langle v, a_i \rangle|^2 \right) \\ &= N \cdot \left(\| v \|_2^{\mathcal{A}} \right)^2 \end{aligned}$$

as required. ■

Fact 7.3.8 *For every two orthonormal bases $\mathcal{A}$ and $\mathcal{B}$:*

$$\| v \|_2^{\mathcal{A}} = \| v \|_2^{\mathcal{B}}$$

Proof: Intuitively, the proof follows from the fact that for *any* orthonormal basis, $\mathcal{A}$, the $L_2^{\mathcal{A}}$ norm of the vector v is equal to the square root of the inner product of v with itself (regardless of the choice of the orthonormal basis $\mathcal{A}$). This is proved formally as follows. Let $\mathcal{A} = \{a_i\}_{i=1}^N$ be an orthonormal basis and let v be a vector. Then there exist a sequence of real numbers $\{\alpha_i\}_{i=1}^N$

(specifically, $\alpha_i = \langle v, a_i \rangle$) so that v can be written as $\sum_{i=1}^{N} \alpha_i \cdot a_i$. By the definition of the $L_2^{\mathcal{A}}$ norm and by the orthonormality of $\mathcal{A}$ we have:

$$\begin{aligned}
\langle v, v \rangle &= \langle \sum_{i=1}^{N} \alpha_i \cdot a_i, \sum_{j=1}^{N} \alpha_j \cdot a_j \rangle \\
&= \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \cdot \alpha_j \cdot \langle a_i, a_j \rangle \\
&= \sum_{i=1}^{N} \alpha_i^2 \qquad (7.8) \\
&= \sum_{i=1}^{N} \langle v, a_i \rangle^2 \\
&= \left(\| v \|_2^{\mathcal{A}} \right)^2
\end{aligned}$$

where Eq. (7.8) follows from the fact that for every $i \neq j$, $\langle a_j, a_i \rangle = 0$ and $\langle a_i, a_i \rangle = 1$. It follows that for every two orthonormal bases $\mathcal{A}$ and $\mathcal{B}$:

$$\left(\| v \|_2^{\mathcal{A}} \right)^2 = \langle v, v \rangle = \left(\| v \|_2^{\mathcal{B}} \right)^2$$

as required. ■

Fact 7.3.9 *For every basis $\mathcal{B}$:*

$$\| v \|_2^{\mathcal{B}} \leq \sqrt{N} \cdot \| v \|_\infty^{\mathcal{B}}$$

Proof: Let $\mathcal{B} = \{b_i\}_{i=1}^{N}$ be a basis. By definition of the $L_2^{\mathcal{B}}$ and $L_\infty^{\mathcal{B}}$ norms:

$$\begin{aligned}
\left(\| v \|_2^{\mathcal{B}} \right)^2 &= \sum_{i=1}^{N} |\langle v, b_i \rangle|^2 \\
&\leq N \cdot \max_{i \leq i \leq N} \{ |\langle v, b_i \rangle|^2 \} \\
&= N \cdot \left(\max_{i \leq i \leq N} \{ |\langle v, b \rangle| \} \right)^2 \\
&= N \cdot \left(\| v \|_\infty^{\mathcal{B}} \right)^2
\end{aligned}$$

as required. ■

By combining Facts 7.3.7, 7.3.8 and 7.3.9 we obtain:

$$\begin{aligned}
\| v \|_1^{\mathcal{A}} &\leq \sqrt{N} \cdot \| v \|_2^{\mathcal{A}} \\
&= \sqrt{N} \cdot \| v \|_2^{\mathcal{B}} \\
&\leq N \cdot \| v \|_\infty^{\mathcal{B}}
\end{aligned}$$

which completes the proof of Lemma 7.3.6.

7.4 (ϵ, k)-Approximations of the Uniform Distribution

7.4.1 (ϵ, k)-approximation of the uniform distribution

In this section, we introduce the notion of "approximately" uniform distributions and show how they can be derived from ϵ-bias sample spaces. Intuitively, we say that a distribution X is an ϵ-approximation of U_n if $\Delta(X, U_n) \leq \epsilon$. We combine this ϵ-approximation with the k-wise independent relaxation that we have seen in previous lectures. That is, we say that X is an (ϵ, k)-approximation of U_n if, when projecting X on any subset of size k of it coordinates, the resulting distribution is an ϵ-approximation of U_k. Formally,

Definition 7.7 ((ϵ, k)-approximation of the uniform distribution): *Let $X \in \{0,1\}^n$ be a random variable, let $\epsilon > 0$ and let $k \in [n]$ be an integer. We say that $X = (X_1, \ldots, X_n)$ is an (ϵ, k)-*approximation *of the uniform distribution, if for every subset $I = \{i_1, \ldots, i_k\} \subseteq [n]$ (of size k) the distribution $X_I = \{X_{i_1}, \ldots, X_{i_k}\}$ is ϵ-far from uniform (that is, $\Delta(X_I, U_k) \leq \epsilon$).*

Clearly, an $(0, k)$-approximation of the uniform distribution is a k-wise independent sample space. Furthermore, an (ϵ, n)-approximation of the uniform distribution is what we informally called an ϵ-approximation above. It is easily verified (see Proposition 7.4.1 below) that for every $\epsilon > 0$ and every $k > 1$, an (ϵ, k)-approximation of the uniform distribution is also $(\epsilon, k-1)$-approximation of the uniform distribution. Essentially, an (ϵ, k)-approximation of uniform is a relaxation of k-wise independence. As we shall see, this further approximation enables us to construct sample spaces that are far smaller than k-wise independent spaces.

Proposition 7.4.1 *For every $\epsilon > 0$ and every $k > 1$, an (ϵ, k)-approximation of the uniform distribution is also an $(\epsilon, k-1)$-approximation of the uniform distribution.*

Proof: Let $X \in \{0,1\}^n$ be a random variable, let $\epsilon > 0$, let $k > 1$ be an integer and suppose that X is an (ϵ, k)-approximation of the uniform distribution. If X is not an $(\epsilon, k-1)$-approximation of the uniform distribution, then there exists a set $I \subseteq [n]$ of size $k-1$ (for simplicity, assume that $I = \{1, \ldots, k-1\}$) so that the distribution $X_I = \{X_1, \ldots, X_{k-1}\}$ is not ϵ-far from U_{k-1} (that is, $\Delta(X_I, U_{k-1}) > \epsilon$). By the alternative definition of statistical difference (Definition 7.2) this means that there exists a set $S \subseteq \{0,1\}^{k-1}$ so that:

$$\Pr[X_I \in S] - \Pr[U_{k-1} \in S] > \epsilon \tag{7.9}$$

For $b \in \{0,1\}$ consider now the set $S_b \subseteq \{0,1\}^k$ that is obtained by concatenating each element $\alpha \in S$ with the bit b (that is, $S_b = \{\alpha \circ b \mid \alpha \in S\}$). Notice that $\Pr[U_k \in S_0 \cup S_1] = \Pr[U_{k-1} \in S]$. Similarly, $\Pr\left[X_{I\cup\{k\}} \in S_0 \cup S_1\right] = \Pr[X_I \in S]$ (where $X_{I\cup\{k\}} = \{X_1, \ldots, X_k\}$ is the distribution that is induced by the k-sized subset $I \cup \{k\}$). Thus, for the set $I \cup \{k\} \subseteq [n]$ of size k, we have:

$$\begin{aligned}
\Delta(X_{I\cup\{k\}}, U_k) &= \max_{S\subseteq\{0,1\}^k} \left\{\Pr\left[X_{I\cup\{k\}} \in S\right] - \Pr[U_k \in S]\right\} \\
&\geq \Pr\left[X_{I\cup\{k\}} \in S_0 \cup S_1\right] - \Pr[U_k \in S_0 \cup S_1] \\
&= \Pr[X_I \in S] - \Pr[U_{k-1} \in S]
\end{aligned}$$

which by Eq. (7.9) is greater than ϵ. This is in contradiction to the assumption that X is an (ϵ, k)-approximation of the uniform distribution. ∎

7.4.2 Achieving (ϵ, k)-approximations of uniform

The following corollary of Theorem 7.5 establishes the relation between the max-bias of X and the quality of the (ϵ, k)-approximation of the uniform distribution achieved by X.

Corollary 7.8 *Let $X \in \{0,1\}^n$ be a random variable such that $\mathrm{maxbias}(X) \le \epsilon'$. Then for every $k \in [n]$, X is an (ϵ, k)-approximation of the uniform distribution, where $\epsilon \stackrel{\text{def}}{=} \frac{\sqrt{2^k}}{2} \cdot \epsilon'$.*

Proof: Let $I \subseteq [n]$ be any subset of indices of size k; denote $I = \{i_1, \ldots, i_k\}$. Then, as with k-wise independence, consider the distribution $X_I = \{X_{i_1}, \ldots, X_{i_k}\}$ (as induced by I). Note that, by definition, it holds that $\mathrm{maxbias}(X_I) \le \mathrm{maxbias}(X)$, implying that $\mathrm{maxbias}(X_I) \le \epsilon'$. Then, by applying Theorem 7.5 we have that

$$2 \cdot \Delta(X_I, U_k) \le \sqrt{2^k} \cdot \mathrm{maxbias}(X_I) \le \sqrt{2^k} \cdot \epsilon'$$

Therefore, for every $I \subseteq [n]$ of size k, the distribution X_I is $\frac{\sqrt{2^k}}{2} \cdot \epsilon'$-far from uniform. ■

Corollary 7.8 implies that in order to obtain (ϵ, k)-approximations of uniform, it is enough to construct an ϵ'-bias sample space (for the appropriate ϵ'). In the next lecture, such a construction will be presented. That is, the following theorem will be proven:

Theorem 7.9 *There exist ϵ-bias sample spaces of size $\mathrm{poly}(n/\epsilon)$. Moreover, such sample spaces can be constructed in $\mathrm{poly}(n/\epsilon)$-time.*

Combining Corollary 7.8 and Theorem 7.9, we obtain:

Corollary 7.10 *For every $\epsilon > 0$ and $k \le n$, one can construct in $\mathrm{poly}(n2^k/\epsilon)$-time an (ϵ, k)-approximation of the uniform distribution over $\{0,1\}^n$.*

Recall that k-wise independent spaces must be of size at least $\Omega(n^{\lfloor k/2 \rfloor})$. Therefore, efficiently constructible spaces of size $\mathrm{poly}(n)$ can only be k-wise independent for *constant* values of k. On the other hand, Corollary 7.10 yields efficiently constructible (ϵ, k)-approximations of uniform, with size just $\mathrm{poly}(n \cdot 2^k/\epsilon)$. An immediation ramification of this is that, unlike for k-wise independence, efficiently constructible polynomial-size $(\frac{1}{\mathrm{poly}(n)}, k)$-approximations of uniform exist for *non-constant* values of k (i.e., for $k = O(\log n)$).

We comment that one can improve over Corollary 7.10 by combining Theorem 7.9 with some known k-wise independent constructions, taking advantage on the linearity of the latter. Details are given in the Appendix.

7.4.3 An application of (ϵ, k)-approximations of uniform

We conclude by briefly showing an application of (ϵ, k)-approximations of the uniform distribution to the problem of kCNF. Suppose that we would like to construct a deterministic algorithm, that given a kCNF formula φ as input finds an assignment that satisfies at least $1 - 2^{-k}$ of the clauses of φ. As we have already seen in Lecture 3, one can use a 3-wise independent sample space in order to solve this problem for $k = 3$ (i.e., in the case of 3CNF).

Specifically, we saw a deterministic algorithm that receives a 3CNF formula φ as input and finds an assignment satisfying at least 7/8 of the clauses of φ. The algorithm described worked by first constructing a 3-wise independent sample space (of size polynomial in the length of φ). The key observation was that for any clause, the probability that it is satisfied by a random element of this sample space *equals* the probability that it is satisfied by a uniformly chosen assignment. Thus, the probability that a given clause is satisfied is exactly 7/8. (The derandomization step consisted of simply traversing the entire 3-wise independent sample space and checking the number of clauses satisfied by each assignment from the space.)

By using a k-wise independent sample space (rather than a 3-wise independent space) one can generalize this approach to construct an algorithm that, given a kCNF formula φ, finds an assignment that satisfies at least $1 - 2^{-k}$ of the clauses of φ. The problem is that the size of any k-wise independent space must be at least $\Omega(n^{\lfloor k/2 \rfloor})$. Therefore, the resulting algorithm is polynomial-time only for constant values of k and also in that case its running time is $\Omega(n^{\lfloor k/2 \rfloor})$ (which is more than the alternative presented below). However, by replacing the k-wise independent space with an appropriately chosen (ϵ, k)-approximation of the uniform distribution one can do significantly better. In particular, one can construct a polynomial-time algorithm that generates the required assignment even if k is non-constant (specifically, it will work as long as $k = O(\log n)$). Details follow.

When using an (ϵ, k)-approximation of the uniform distribution (instead of a k-wise independent sample space), the probability that any given clause is satisfied is at least $(1 - 2^{-k}) - \epsilon$, and the expected number of clauses that are satisfied is at least $(1 - 2^{-k}) \cdot n - \epsilon \cdot n$ (where n equals the number of clauses in φ). Now, if we set ϵ such that $\epsilon \cdot n < 2^{-k}$ (e.g., $\epsilon = 1/(10 \cdot 2^k \cdot n)$), then we have that there exists a satisfying assignment in the sample space satisfying $1 - 2^{-k}$ of the clauses. By Corollary 7.8, we have that an (ϵ, k)-approximation of the uniform distribution can be obtained from a $(2 \cdot \epsilon/\sqrt{2^k})$-bias space. Thus, for $\epsilon = 1/(10 \cdot 2^k \cdot n)$ we need to construct (and scan) an ϵ'-bias sample space with $\epsilon' = 2/(10 \cdot 2^{3 \cdot k/2} \cdot n)$. By Theorem 7.9 this can be done in time $\text{poly}(n/\epsilon') = \text{poly}(n, 2^k)$, giving us the desired result. We note that $O((n \cdot 2^k)^c)$ is preferrable to $O(n^{\lfloor k/2 \rfloor})$ for every $k > 2 \cdot c$ and sufficiently large n.

Bibliographic Notes

Small-bias sample spaces were introduced by Naor and Naor [35]. The relation between statistical difference and the maximum bias of XORs of bits was first discovered by Vazirani [43], but the proof presented here follows the one given in the appendix of [8].

Appendix

> Oded's Note: *The following material was not presented in class (although this was the original plan). This, as well as other misfortunes, is due to the slower than expected pace I was forced to use.*

We present an improvement over Corollary 7.10. Recall that Corollary 7.10 was proven by combining the generic relation between small-bias and statistical-distance (as applied to "windows of size k" of the distribution) with a construction of small-bias sample spaces (i.e., Theorem 7.9). Here we combining Theorem 7.9 with some known k-wise independent constructions, taking advantage of their linearity. We start with the transformation.

Lemma (Naor and Naor [35]): *Let $S_n \subseteq \{0,1\}^n$ be an ϵ-bias sample space. Let $k \leq N$ and $L_{k,N}^n \subseteq \{0,1\}^N$ be a k-wise independent sample space of cardinality 2^n. Suppose $L_{k,N}^n$ is defined by the linear mapping $T : \{0,1\}^n \to \{0,1\}^N$* (i.e, $T(x \oplus y) = T(x) \oplus T(y)$). *Then, the sample* *space $R_{k,N}$ constructed by applying the linear map T to each sample point in S_n, has bias at most ϵ relative to any non-empty subset of size at most k. Hence, $R_{k,N}$ is a $(2^{k/2}\cdot\epsilon, k)$-approximation of the uniform distribution over $\{0,1\}^N$.*

Indeed, $|R_{k,N}| = |S_n|$, where n is such that a k-wise independent sample space (over $\{0,1\}^N$) as required exists. We will see that such spaces exist for $n = k\log_2 N$, and so we will get an $(2^{k/2}\cdot\epsilon, k)$-approximation (of the uniform distribution) over $\{0,1\}^N$ using a sample space of size $\text{poly}(n/\epsilon) = \text{poly}((k\log N)/\epsilon)$, improving over Corollary 7.10 (which yields size $\text{poly}(N/\epsilon)$ for such spaces).

Proof: Let $X = X_1\cdots X_N$ denote a random variable that is uniformly distributed in $R_{k,N} \subseteq \{0,1\}^N$. By the construction, $X = T(Y)$, where $Y = Y_1\cdots Y_n$ denote a random variable that is uniformly distributed in $S_n \subseteq \{0,1\}^n$. Our hypothesis is that Y is ϵ-bias, and our first goal is to bound the bias of X relative to *any non-empty subset I of size at most k*. Let $M = (a_{i,j})$ denote the N-by-n Boolean matrix effecting the linear map T (i.e., $T(Y) = MY$). We have

$$\begin{aligned}
\Pr\left[\bigoplus_{i\in I} X_i = 0\right] &= \Pr\left[\bigoplus_{i\in I} T(Y)_i = 0\right] \\
&= \Pr\left[\bigoplus_{i\in I}\left(\bigoplus_{j=1}^{n} a_{i,j}Y_j\right) = 0\right] \\
&= \Pr\left[\bigoplus_{j=1}^{n}\bigoplus_{i\in I} a_{i,j}Y_j = 0\right] \\
&= \Pr\left[\bigoplus_{j\in J} Y_j = 0\right]
\end{aligned}$$

where $j \in J$ if and only if $\bigoplus_{i\in I} a_{i,j} = 1$. Thus, $\text{bias}_I(X) = \text{bias}_J(Y)$, which is bounded by ϵ, provided that $J \neq \emptyset$ (because Y is ϵ-bias). But $J \neq \emptyset$ must hold, because otherwise $\text{bias}_I(T(U_n)) = \text{bias}_\emptyset(U_n) = 1$ (where U_n is the uniform distribution over $\{0,1\}^n$) in contradiction to the hypothesis that $T(U_n)$ is k-wise independent (which implies that $\text{bias}_I(T(U_n)) = 0$ for every non-empty subset of size at most k, such as I). The establishes the main claim. The second claim follows by applying Corollary 7.8. ■

Constructing k-wise independent sample spaces via linear maps: It is left to show that we can construct k-wise independent sample spaces (of the uniform distribution) over $\{0,1\}^N$ by using linear maps from $\{0,1\}^n$, where $n = k\log_2 N$. Recall that the sample spaces presented in Lecture 3 have the desired size (i.e., $2^n = N^k$), but are they linear? These sample spaces were described in terms of polynomial of degree $k-1$ over $F \stackrel{\text{def}}{=} GF(2^\ell)$, where $\ell = \lceil\log_2 N\rceil$, and are naturally presented as a mapping from F^k to F^N. Specifically, a degree $k-1$ polynomial represented by the coefficient sequence $(c_0, ..., c_{k-1}) \in F^k$ is mapped to the sequence of values $(v_1, ..., v_N) \in F^N$, where $v_j = \sum_{i=0}^{k-1} c_i\alpha_j^i$ and α_j is the jth element of the field F. Noting that here the α_j's are fixed, we get linear maps over F:

$$(c_0, ..., c_{k-1}) \mapsto \sum_{i=0}^{k-1} \alpha_j^i c_j$$

Recalling that $F = GF(2^\ell)$, these linear maps over F can be written as linear maps over $GF(2)$: elements of $F = GF(2^\ell)$ are viewed as polynomials of degree $\ell-1$, and multiplication by fixed elements (i.e., the powers of the α_j's) can be expressed as linear combinations of the $GF(2)$-coefficients of these polynomials. (For further details, consult any standard Coding Theory book; for example, [32, Chap. 4].)

Lecture 8

Small Bias Sample Spaces (Part 2)

Notes taken by Boaz Barak and Itsik Mantin

Summary: In the previous lecture we discussed the definition of small bias sample spaces and some properties of such spaces. In this lecture we prove that such sample spaces can be constructed in polynomial time, and demonstrate an application of this construction in proving a tight hardness of approximation result for an $\mathcal{NP}$-complete problem.

8.1 Introduction

In the previous lecture, we have seen the definition of ϵ-bias sample spaces. Let us recall this definition:

Definition 8.1 *Let $n \in \mathsf{N}$ and $\epsilon > 0$. A (multi) set $\Omega \subseteq \{0,1\}^n$ is called an* ϵ-bias sample space over $\{0,1\}^n$ *if for any non-empty $I \subseteq \{1,\ldots,n\}$*

$$\left|\Pr_{\mathbf{s}\in\Omega}[\bigoplus_{i\in I} s_i = 0] - \Pr_{\mathbf{s}\in\Omega}[\bigoplus_{i\in I} s_i = 1]\right| < \epsilon$$

where $s = s_1 \ldots s_n$.

An equivalent formulation is that for any non-empty $I \subseteq \{1,\ldots,n\}$ it holds that

$$\Pr_{\mathbf{s}\in\Omega}[\bigoplus_{i\in I} s_i = 0] \in \frac{1}{2} \pm \frac{\epsilon}{2}$$

(where by $a \pm c$ we mean the interval $(a-c, a+c)$)

Last lecture we noted that the uniform distribution has bias zero, and proved a relation between the bias of a distribution and its statistical difference from the uniform distribution.

This lecture we will see (in Section 8.2) that ϵ-bias sample spaces can be constructed in time polynomial in $\frac{n}{\epsilon}$. In particular this ϵ-bias sample spaces (over $\{0,1\}^n$) has size $poly(\frac{n}{\epsilon})$ (as opposed to the uniform sample space over $\{0,1\}^n$ that is of size 2^n).

In Section 8.3 we will use this construction to prove that a specific $\mathcal{NP}$-complete problem (Quadratic Equations) that can be approximated with ratio $\frac{1}{2}$ cannot be approximated with any better constant ratio, unless $\mathcal{P} = \mathcal{NP}$. This is one of the few tight hardness of approximation results that can be obtained without using the $\mathcal{PCP}$ Theorem.

8.2 Construction of a Small Bias Sample Space

In this section we will show that it is possible to construct an ϵ-bias sample space of size $poly(\frac{n}{\epsilon})$. That is, we prove the following Theorem:

Theorem 8.2 *There exists an algorithm that, on input n and ϵ outputs a list of n-bit strings that constitutes an ϵ-bias space and does so within time $poly(\frac{n}{\epsilon})$.*

8.2.1 On the Existence of $poly(\frac{n}{\epsilon})$-size ϵ-bias sample spaces.

Before proving Theorem 8.2 we note that the mere *existence* of an ϵ-bias sample space of $poly(\frac{n}{\epsilon})$ size can be easily proved using the probabilistic method. That is, we have the following:

Proposition 8.2.1 *For any $n \in \mathbb{N}$, $\epsilon > 0$, there exists an ϵ-bias sample space over $\{0,1\}^n$ of size $O(\frac{n}{\epsilon^2})$.*

Proof: Let Ω be a (multi) set containing strings in $\{0,1\}^n$. For any non-empty $I \subseteq \{1,\ldots,n\}$ we define $f_I(\Omega)$ to be the number of strings $\mathbf{s} = s_1 \ldots s_n \in \Omega$ such that $\bigoplus_{i\in I} s_i = 0$. By definition, it holds that Ω is an ϵ-bias sample space if (and only if) for any such I, it holds that $f_I(\Omega) \in (\frac{1}{2} \pm \frac{\epsilon}{2}) \cdot |\Omega|$.

Consider a random (multi) set Ω of size $\frac{n}{\epsilon^2}$. That is $\Omega = \{\mathbf{s}^{(1)}, \ldots, \mathbf{s}^{(n/\epsilon^2)}\}$ where for each j, $\mathbf{s}^{(j)}$ is chosen uniformly and independently in $\{0,1\}^n$. Let us fix I to be a non-empty subset of $\{1,\ldots,n\}$, and let X_I be the random variable $f_I(\Omega)$. Another way to define X_I is as $\sum_{j=1}^{|\Omega|} Y_j$ where Y_j is a random variable that is defined to be 1 if $\oplus_{i\in I}\mathbf{s}_i^{(j)} = 0$ and 0 otherwise. The fact that the uniform distribution has bias zero implies that $\Pr[Y_j = 1] = \frac{1}{2}$, and so $\mu \stackrel{\text{def}}{=} Ex[X] = \frac{|\Omega|}{2} = \frac{n}{2\epsilon^2}$.

The random variables $Y_1, \ldots, Y_{k/\epsilon^2}$ are independent. Therefore, we can apply the Chernoff bound and get that (for every $I \neq \emptyset$)

$$\begin{aligned}
\Pr_\Omega[f_I(\Omega) \notin (\frac{1}{2} \pm \frac{\epsilon}{2}) \cdot |\Omega|] &= \Pr_\Omega[|X_I - \mu| > \epsilon\mu] \\
&= \Pr_\Omega[|\sum_{j=1}^{|\Omega|} Y_j - \mu| > \epsilon\mu] \\
&< 2e^{-2\epsilon^2\mu} \\
&= 2e^{-2\epsilon^2\cdot(n/2\epsilon^2)} \\
&= 2e^{-n}
\end{aligned}$$

For sufficiently large n, this probability is *strictly smaller* than 2^{-n}. As there are at most 2^n subsets of $\{1,\ldots,n\}$, we get that by the union bound there is a *positive probability* that for a random Ω of size $\frac{n}{\epsilon^2}$, for *any* non-empty set $I \subseteq \{1,\ldots,n\}$ the number $f_I(\Omega)$ is in $(\frac{1}{2} \pm \frac{1}{2}\epsilon)\cdot|\Omega|$. In particular there must *exist* at least one (multi) set Ω of size $\frac{n}{\epsilon^2}$ that satisfies the above condition for any non-empty $I \subseteq \{1,\ldots,n\}$. Such a set Ω will be an ϵ-bias sample space.

■

8.2.2 The Construction

We will prove Theorem 8.2 by showing a polynomial-time algorithm that outputs an ϵ-bias sample space Ω of size $O((\frac{n}{\epsilon})^2)$. This sample space will be larger than the sample space of Proposition 8.2.1 (which is $O(\frac{n}{\epsilon^2})$) but it will be explicitly constructible. We will start by showing the construction and then provide its analysis.

LFSR sequences

The primary tool we will use is a *Linear Feedback Shift Register* (LFSR). An LFSR is a sequence of bits defined by two parameters: a *starting sequence* and a *feedback rule*. The name linear feedback shift register comes from a popular hardware implementation of this sequence.

Notation. In this section it will be convenient for us to treat strings as indexed from 0 to $l-1$ (where l is the string's length) instead of being indexed from 1 to l.

Definition 8.3 *Let $l \in \mathbb{N}$ and let $\mathbf{s} = s_0 \cdots s_{l-1}$ and $\mathbf{f} = f_0 \cdots f_{l-1}$ be two l-bit strings. The* LFSR sequence defined by $\mathbf{s}$ and $\mathbf{f}$ *is the sequence $(r_0, r_1, r_2, \ldots)$ defined as follows:*

$$r_i \stackrel{\text{def}}{=} \begin{cases} s_i & 0 \le i \le l-1 \\ \sum_{j=0}^{l-1} f_j \cdot r_{(i-l)+j} & i \ge l \end{cases}$$

We call $\mathbf{s}$ *the* starting sequence *and* $\mathbf{f}$ *the* feedback rule.

Note that the definition of an LFSR sequence is a recursive definition, with each bit r_i (for $i \ge l$) depending only on the previous l bits. Therefore, this sequence will repeat itself after at most 2^l steps, or in other words, it is a periodic sequence with period at most 2^l.

Let us fix $l, n \in \mathbb{N}$ with $l < n$ (actually we will use $l \ll n$). We define the function $\mathrm{seq}_n : \{0,1\}^l \times \{0,1\}^l \to \{0,1\}^n$ as follows: for $\mathbf{s}, \mathbf{f} \in \{0,1\}^l$, $\mathrm{seq}_n(\mathbf{s}, \mathbf{f})$ consists of the first n bits of the LFSR sequence defined by $\mathbf{s}$ and $\mathbf{f}$. That is $\mathrm{seq}_n(\mathbf{s}, \mathbf{f}) = r_0 \cdots r_{n-1}$, where $r_0, \ldots, r_{n-1}$ are defined as in Definition 8.3.

Polynomials over $GF(2)$. To describe the rest of the construction, we will need to use polynomials with coefficients in $GF(2)$. Recall that one can define multiplication and division for such polynomials. We say that a polynomial is *irreducible* if it has no non-trivial factors (i.e., the only polynomials that divide it are itself and the polynomial $f(t) \equiv 1$). That is, irreducible polynomials in the polynomial ring correspond to prime numbers in the domain of natural numbers.

It will be useful for us to identify the feedback rule $\mathbf{f} = (f_0, \ldots, f_{l-1})$ with the following l-degree polynomial with coefficients in $GF(2)$:

$$f(t) = -t^l + \sum_{j=0}^{l-1} f_j \cdot t^j$$

(Note that $-1 = +1$ over $GF(2)$. The reason that we prefer to write the definition of the polynomial $f(t)$ in the above way, will become clear later on)

We say that $\mathbf{f}$ is a *valid* feedback rule if the corresponding polynomial is *irreducible* over $GF(2)$. We let $IP_l \subseteq \{0,1\}^l$ denote the set of all valid feedback rules (or equivalently, the set of all irreducible polynomials of degree exactly l over $GF(2)$). The following fact is well-known:

Fact 8.2.2 $|IP_l| = \Theta(\frac{2^l}{l})$.

The actual sample space

We define the following sample space

$$\Omega_{n,l} \stackrel{\text{def}}{=} \{\text{seq}_n(\mathbf{f},\mathbf{s}) : \mathbf{f} \in \mathcal{IP}_l, \mathbf{s} \in \{0,1\}^l\}$$

We have that $|\Omega_{n,l}| = \Theta(\frac{2^{2l}}{l})$. Moreover, it is possible to enumerate the elements of $\Omega_{n,l}$ in time polynomial in n and 2^l. Indeed, given $\mathbf{s},\mathbf{f} \in \{0,1\}^l$ the function $\text{seq}_n(\mathbf{s},\mathbf{f})$ can be evaluated in $poly(n)$ time, and enumerating all possible choices for $\mathbf{s},\mathbf{f}$ can be done in $2^{O(l)}$ time (note that checking whether a polynomial of degree l over $GF(2)$ is irreducible can be done trivially in time $2^{O(l)}$ by checking all possible factors

8.2.3 Analysis

We prove Theorem 8.2 by proving the following proposition:

Proposition 8.2.3 *For some constant k and for every $l,n \in \mathsf{N}$ with $l < n$, the set $\Omega_{n,l}$ is a $k \cdot \frac{n}{2^l}$-bias sample space over $\{0,1\}^n$.*

Proposition 8.2.3 immediately implies Theorem 8.2. This is because if we let $l = \log(k \cdot \frac{n}{\epsilon})$, then by Proposition 8.2.3, $\Omega_{n,l}$ is an ϵ-bias sample space over $\{0,1\}^n$ of size $O(\frac{2^{2l}}{l}) = O((\frac{n}{\epsilon})^2)$ that is constructible in $poly(\frac{n}{\epsilon})$-time.

We will start by some preliminary observations on the connection between the LFSR sequence and the polynomial that corresponds to the feedback rule. Then, in Section 8.2.3, we shall use these observations to prove Proposition 8.2.3.

Preliminary Observations

Let us fix $l < n$ to be two natural numbers as in the statement of Proposition 8.2.3. Let $\mathbf{s}$ and $\mathbf{f}$ be a starting sequence and a feedback rule in $\{0,1\}^l$ and let $(r_0,\ldots,r_{n-1}) = \text{seq}_n(\mathbf{s},\mathbf{f})$. We know that for any $i \geq l$

$$r_i = \sum_{j=0}^{l-1} f_j \cdot r_{(i-l)+j} \tag{8.1}$$

It is clear that any value r_i is a linear combination of the value of the starting sequence $s_0,\ldots,s_{l-1}$. That is, there exists an $n \times l$ matrix $C = (c_{i,j})$ over $GF(2)$ such that for any i

$$r_i = \sum_{j=0}^{l-1} c_{i,j} \cdot s_j \tag{8.2}$$

In matrix notation, we can write $C\mathbf{s} = \mathbf{r}$, where $\mathbf{r}$ is the vector $\text{seq}_n(\mathbf{s},\mathbf{f}) = (r_0,\ldots,r_{n-1})$. The entries of the matrix C can be obtained by repeated applications of Equation 8.1, and hence are determined by the value of the feedback rule $\mathbf{f}$. We will sometimes denote this matrix C by $C(\mathbf{f})$.

Consider the polynomial $f(t)$ of degree l that corresponds to $\mathbf{f}$ (that is, $f(t) = -t^l + \sum_{j=0}^{l-1} f_j \cdot t^j$). We can reduce any polynomial (of degree at least l) modulo this polynomial $f(t)$. In particular we have that

$$t^l = \sum_{j=0}^{l-1} f_j \cdot t^j - f(t) \equiv \sum_{j=0}^{l-1} f_j \cdot t^j \pmod{f(t)}$$

For $i \geq l$ we have that

$$t^i = t^{i-l}t^l \equiv t^{i-l}(\sum_{j=0}^{l-1} f_j t^j) \equiv \sum_{j=0}^{l-1} f_j t^{(i-l)+j} \pmod{f(t)} \tag{8.3}$$

Note the analogy between Equations 8.1 and 8.3. The polynomial t^i is equivalent (modulo $f(t)$) to a polynomial of degree $l-1$, or in other words to a linear combination of $t^0, \ldots, t^{l-1}$. The coefficients of this linear combination can be obtained by repeated application of Equation 8.3. The analogy between Equations 8.1 and 8.3 implies that these will be the *same coefficients* as in Equation 8.2. That is, the following holds:

$$t^i \equiv \sum_{j=0}^{l-1} c_{i,j} \cdot t^j \pmod{f(t)} \tag{8.4}$$

where $C = (c_{i,j})$ is the very same $n \times l$ matrix $C(\mathbf{f})$ defined above. In matrix notation we have that

$$C \begin{pmatrix} 1 \\ t \\ \vdots \\ t^{l-1} \end{pmatrix} \equiv \begin{pmatrix} 1 \\ t \\ \vdots \\ \vdots \\ t^{n-1} \end{pmatrix} \pmod{f(t)} \tag{8.5}$$

By applying the transpose operation to Equation 8.5, we obtain the following equation that will be useful for us later on:

$$(1 \cdots t^{l-1})C^T \equiv (1 \cdots t^{n-1}) \pmod{f(t)} \tag{8.6}$$

Proving Proposition 8.2.3

In order to prove that $\Omega_{n,l}$ is a $O(\frac{n}{2^l})$-bias sample space we need to show that the following holds for any non-empty $I \subseteq \{0, \ldots, n-1\}$:

$$\Pr_{\mathbf{r} \in \Omega_{n,l}} \left[\bigoplus_{i \in I} r_i = 0 \right] \in \frac{1}{2} \pm O(\frac{n}{2^l})$$

Equivalently, we need to prove that for any non-zero vector $\mathbf{v} \in \{0,1\}^n$

$$\Pr_{\mathbf{r} \in \Omega_{n,l}} [\langle \mathbf{v} \, , \, \mathbf{r} \rangle = 0] \in \frac{1}{2} \pm O(\frac{n}{2^l}) \tag{8.7}$$

where $\langle \cdot \, , \, \cdot \rangle$ denotes the standard dot-product (i.e., $\langle \mathbf{v} \, , \, \mathbf{r} \rangle \stackrel{\text{def}}{=} \mathbf{v}^T \mathbf{r} = \sum_{i=0}^{n-1} u_i \cdot r_i \pmod 2$). In this notation, the fact that the uniform distribution has bias 0 can be phrased as follows:

Fact 8.2.4 *Let $\mathbf{v} \neq 0$ be a vector in $\{0,1\}^n$, then*

$$\Pr_{\mathbf{r} \in \{0,1\}^n} [\langle \mathbf{v} \, , \, \mathbf{r} \rangle = 0] = \frac{1}{2}$$

Let us fix a non-zero vector $\mathbf{v} \in \{0,1\}^n$ and let p denote the probability defined in Equation 8.7. By the definition of the set $\Omega_{n,l}$ we know that

$$\begin{aligned} p &= \Pr_{\mathbf{s}\in\{0,1\}^l, \mathbf{f}\in IP_l}[\langle \mathbf{v}\,,\, \mathrm{seq}_n(\mathbf{s},\mathbf{f})\rangle = 0] \\ &= \Pr_{\mathbf{s}\in\{0,1\}^l, \mathbf{f}\in IP_l}[\langle \mathbf{v}\,,\, C(\mathbf{f})\mathbf{s}\rangle = 0] \\ &= \Pr_{\mathbf{s}\in\{0,1\}^l, \mathbf{f}\in IP_l}[\langle C(\mathbf{f})^T\mathbf{v}\,,\, \mathbf{s}\rangle = 0] \end{aligned}$$

where the last equality holds by a well-known property of the dot product. Yet, by Fact 8.2.4 above, we know that, conditioned on $C(\mathbf{f})^T\mathbf{v}$ being a non-zero vector, the above probability is exactly half. We also know that if $C(\mathbf{f})^T\mathbf{v}$ is the zero vector, then we will have that $\langle C(\mathbf{f})^T\mathbf{v}\,,\, \mathbf{s}\rangle = 0$ with probability 1. It follows that if we let A denote the event that $C(\mathbf{f})^T\mathbf{v}$ is the zero vector, and let ρ denote the probability (over $\mathbf{f} \in IP_l$) that A happens, then

$$p = (1-\rho)\cdot\frac{1}{2} + \rho\cdot 1 = \frac{1}{2} + \frac{1}{2}\cdot\rho$$

Therefore, to prove Proposition 8.2.3, all we need to do is to prove that $\rho = O(\frac{n}{2^l})$. Consider the case that the event A happens. In this case we have that

$$C(\mathbf{f})^T\mathbf{v} = \mathbf{0}$$

and so (for $C = C(\mathbf{f})$) it holds that for any l-bit vector $\mathbf{x}$

$$\mathbf{x}C^T\mathbf{v} = 0$$

(note that in the first equation $\mathbf{0}$ denotes the zero l-bit vector, while in the second equation we use the scalar 0).

In particular if we plug $\mathbf{x} = (1\cdots t^{l-1})$ we will get that the polynomial $(1\cdots t^{l-1})C^T\mathbf{v}$ is the zero polynomial (modulo $f(t)$). Combining this with Equation 8.6 we get that

$$(1\cdots t^{n-1})\mathbf{v} \equiv 0 \pmod{f(t)}$$

which means that the polynomial $f(t)$ divides the non-zero polynomial $g_{\mathbf{v}}(t) \stackrel{\text{def}}{=} (1\cdots t^{n-1})\mathbf{v} = \sum_{i=0}^{n-1} u_i \cdot t^i$. Note that the polynomial $g_{\mathbf{v}}(t)$ depends only on $\mathbf{v}$ and has degree at most n. We have that $\rho \leq \Pr_{\mathbf{f}\in IP_l}[f(t) \text{ divides } g_{\mathbf{v}}(t)]$. Yet any polynomial of degree at most n has at most $\frac{n}{l}$ *irreducible*[1] factors of degree l, and so the probability that $f(t)$ happens to be one of these factors is at most $\frac{n/l}{|IP_l|}$, which by Fact 8.2.2 is equal to $\frac{n/l}{\Theta(2^l/l)} = O(\frac{n}{2^l})$.

■

8.3 Using Small Size Sample Spaces for proving Hardness of MaxQE

In this section we present the problem of Quadratic Equations satisfiablity (QE). We state that the QE problem is $\mathcal{NP}$-Complete (and sketch the proof) and discuss the approximability of the

[1]Note that because different *reducible* polynomials may have shared factors, it may be the case that a polynomial of degree n has much more than n/l factors of degree l that are *not* irreducible. Therefore it is crucial that we limit ourselves only to valid feedback rules (i.e., feedback rules that correspond to irreducible polynomials).

optimization variant, denoted as MaxQE. We state that this problem can be approximated with a ratio of one half, and we use a construction of small bias sample spaces (e.g., from Theorem 8.2), to prove that this bound is tight and no polynomial algorithm can approximate MxaQE with a better constant ratio (unless $\mathcal{P} = \mathcal{NP}$).

8.3.1 The problem of Quadratic Equations

A quadratic expression is a multi-variate polynomial with maximal degree of 2. A quadratic equation is an equation whose left side is a quadratic expression, and whose right side is a scalar. Given a set of quadratic equations sharing the same variables, a question that arise is whether there exists an assignment for this variables, which satisfies all equations, or formally:

Definition 8.4 *The set* Quadratic Equations *denoted QE, consists of all mutually satisfiable sequences of quadratic equations over $GF(2)$.*

Since in $GF(2)$ it holds that $x = x^2$ for any number x, we can assume without loss of generality that all the summands are either of degree 2 or are constants. The formulation of QE as a decisional problem is given by:

Input A sequence of n quadratic equations with t variables over $GF(2)$

Question Is there an assignment to the variables that satisfies *all* the equations.

Theorem 8.5 $QE \in \mathcal{NPC}$

Proof Sketch: Clearly QE is in $\mathcal{NP}$ as a satisfying assignment is an easily verifiable witness. To see that it is $\mathcal{NP}$*-hard* we will see a reduction from $3SAT$.

Consider an instance of $3SAT$, denoted $\varphi = \bigwedge_{i=1}^{m} C_i$, where each clause C_i is the disjunction of three literals l_{i1}, l_{i2} and l_{i3}, each of them being either one of the variables $p_1, \ldots, p_n$ or its negation.

The reduction consists of two stages, the first one translating the formula into a sequence of m cubic equations, while the second translates this sequence into a sequence of quadratic equations.

The first translation is natural, as $3CNF$ formulas have immediate correspondence to cubic equations. For example, a Boolean equation of the form $p_1 \vee p_2 \vee \neg p_3 \stackrel{?}{=} TRUE$ can be translated by De-Morgan's rules into $\neg p_1 \wedge \neg p_2 \wedge p_3 \stackrel{?}{=} FALSE$, which corresponds to the cubic equation $(1 - x_1)(1 - x_2)x_3 \stackrel{?}{=} 0$.

It is clear that the 3SAT input φ is satisfiable if and only if the sequence of cubic equations is satisfiable. Next, we describe the translation of these equations into quadratic equations. For each pair of variables x_i and x_j in the cubic system, we introduce a new variable z_{ij} and a new equation $z_{ij} = x_i x_j$. Then, we go through all the equations and whenever we find a term of degree 3 (e.g., $x_i x_j x_k$) we replace it by a term of degree 2 (e.g., $z_{ij} x_k$) by replacing the multiplication of two variables $x_i x_j$ with the new variable z_{ij}. Our new equation system is composed of two parts, the first consists of the new added equations $\{z_{ij} = x_i x_j\}_{i,j=1}^{n}$ (and thus is quadratic in the size of the original input), and the second part is the transformed sequence of equations all of degree 2 (linear in the size of the input).

Consider for example, the 3SAT input

$$(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_2 \vee x_3 \vee x_4) \wedge (x_1 \vee x_4 \vee x_5) \stackrel{?}{=} TRUE$$

By applying De-Morgan's rules we get an equivalent formula

$$(\neg x_1 \wedge \neg x_2 \wedge x_3) \vee (x_2 \wedge \neg x_3 \wedge \neg x_4) \vee (\neg x_1 \wedge \neg x_4 \wedge \neg x_5) \stackrel{?}{=} FALSE$$

which is (notice that for the formula to be satisfied, each of the clauses must be FALSE)

$$\begin{pmatrix} \neg x_1 \wedge \neg x_2 \wedge x_3 \\ x_2 \wedge \neg x_3 \wedge \neg x_4 \\ \neg x_1 \wedge \neg x_4 \wedge \neg x_5 \end{pmatrix} \stackrel{?}{=} \begin{pmatrix} FALSE \\ FALSE \\ FALSE \end{pmatrix}$$

This triplet of Boolean equations is translated into the cubic equations

$$\begin{pmatrix} (1-x_1)(1-x_2)x_3 \\ x_2(1-x_3)(1-x_4) \\ (1-x_1)(1-x_4)(1-x_5) \end{pmatrix} \stackrel{?}{=} \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

which is

$$\begin{pmatrix} x_1x_2x_3 - x_1x_3 - x_2x_3 + x_3 \\ x_2x_3x_4 - x_2x_3 - x_2x_4 + x_2 \\ -x_1x_4x_5 + x_4x_5 + x_1x_4 + x_1x_5 - x_4 - x_5 - x_1 + 1 \end{pmatrix} \stackrel{?}{=} \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Notice that all the cubic terms include either the term x_2x_3 or the term x_1x_4, and thus it suffices to replace only these quadratic products by new variables, i.e. z_{23} and z_{24} to reduce the maximal degree into 2

$$\begin{pmatrix} x_1x_4 - z_{14} \\ x_2x_3 - z_{23} \\ x_3 + x_1z_{23} - x_1x_3 - z_{23}) \\ x_2 + z_{23}x_4 - z_{23} - x_2x_4 \\ -z_{14}x_5 + x_4x_5 + x_1x_4 + x_1x_5 - x_4 - x_5 - x_1 + 1 \end{pmatrix} \stackrel{?}{=} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

The last technical step is to go over the equation sequence and apply the following three technical transforms: a) Replace subtractions by additions (its the same in $GF(2)$); b) Replace every linear term by its square; c) Move the scalar '1' to the right side. Finally, we get an input for the QE problem

$$\begin{pmatrix} x_2x_3 + z_{23}^2 \\ x_1x_4 + z_{14}^2 \\ x_3^2 + x_1z_{23} + x_1x_3 + z_{23}^2 \\ x_2^2 + z_{23}x_4 + z_{23}^2 + x_2x_4 \\ z_{14}x_5 + x_4x_5 + x_1x_4 + x_1x_5 + x_4^2 + x_5^2 + x_1^2 \end{pmatrix} \stackrel{?}{=} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

which is satisfiable if and only if the original 3SAT input φ was satisfiable. Since the entire procedure of reducing a $3CNF$ to a sequence of quadratic equations can be done in polynomial time we have reduced $3SAT$ to QE. ■

8.3.2 The Optimization Problem MaxQE

Next, we define the optimization variant of QE, denoted MaxQE:

Input A sequence of n quadratic equations with t variables over $GF(2)$

Problem Find an assignment that satisfies *as many* of the equations *as possible*.

We define the function $MaxQE(I) \in [0, 1]$ to be the maximal fraction of equations in I, which can be mutually satisfied by a single assignment.

Example: The following is an instance for both QE and MaxQE, with the parameters $n = 5$ and $t = 6$.

$$\begin{cases} x_2x_3 + z_{23}^2 = 0 \\ x_1x_4 + z_{14}^2 = 0 \\ x_3^2 + x_1z_{23} + x_1x_3 + z_{23}^2 = 0 \\ x_2^2 + z_{23}x_4 + z_{23}^2 + x_2x_4 = 0 \\ z_{14}x_5 + x_4x_5 + x_1x_4 + x_1x_5 + x_4^2 + x_5^2 + x_1^2 = 1 \end{cases}$$

Notice that the assignment

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ z_{23} \\ z_{14} \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ * \\ 1 \\ 0 \end{pmatrix}$$

satisfies all the equations (there are several satisfying assignments for this QE input), and consequently the the answer to the decisional problem is "YES", whereas the answer for the MaxQE problem is one of the satisfying assignments (and $MaxQE(I) = 1$).

The best known poly-time approximation algorithm for MaxQE gives an approximation ratio of $\frac{1}{2}$. The algorithm that achieves this ratio (and its analysis) are beyond the scope of this course and are omitted.

8.3.3 MaxQE is hard to $(\frac{1}{2} + \epsilon)$-approximate

Theorem 8.6 *For any $\epsilon > 0$ MaxQE is $\mathcal{NP}$-Hard to approximate to within factor $\frac{1}{2} + \epsilon$.*

Before getting to the proof itself, we present new terminology which we use in the proof. We extend the decisional QE problem into a family of problems, denoted $GapQE_{\alpha,\beta}$, parameterized by $\alpha, \beta \in [0, 1]$. For this purpose let us define a promise problem. Loosely speaking, a promise problem is a problem for which the input-output relation is only partially defined, i.e. for some inputs the output is strictly defined (as in typical problems), while for other inputs the output is insignificant.

Next we define two classes of QE inputs

$\forall\alpha$ $YesQE_\alpha$:

A sequence of n quadratic equations with t variables over $GF(2)$ such that there is an assignment to the variables that satisfies at least a fraction α of the equations.

$\forall\beta$ $NoQE_\beta$:

A sequence of n quadratic equations with t variables over $GF(2)$ such that all the assignments to the variables satisfy strictly less than a fraction β of the equations.

Example: $YesQE_1$ is the set of all satisfiable QE inputs, while $NoQE_1$ is the set of all other QE inputs (those which are not satisfiable).

The problem $GapQE_{\alpha,\beta}$ (for $\alpha \geq \beta \in [0, 1]$), is the promise problem of distinguishing $YesQE_\alpha$ inputs from $NoQE_\beta$. However, the fact that $NoQE_\beta$ includes inputs whose best assignments satisfy *strictly less* than a fraction of β, implies that $GapQE_{\alpha,\alpha}$ is defined. Notice that in this case the

output is defined for inputs which are α-satisfiable (and thus in $YesQE_\alpha$) and for inputs which are not α-satisfiable (and thus in $NoQE_\alpha$), i.e. for all inputs. Consequently $GapQE_{\alpha,\alpha}$ is actually a standard decisional problem.
Example: $GapQE_{1,1} \equiv QE$

Lemma 8.3.1 *Suppose that there exists a polynomial time β-approximation algorithm for QE. Then $GapQE_{1,\beta}$ is decidable in polynomial time.*

Proof: For QE inputs from $YesQE_1$, a β-approximation algorithm is promised to provide an assignment satisfying at least β of the equations. For inputs of $NoQE_\beta$ no algorithm can find such an assignment, since no assignment satisfies β of the equations. Thus, applying such an algorithm to some QE input, one can compare the number of satisfied equations to β, and decide about the output for $GapQE_{1,\beta}$ accordingly. ■

The proof of Theorem 8.6 is constructed in the following way: we prove that for every $\epsilon > 0$, $GapQE_{1,\frac{1}{2}+\epsilon}$ is $\mathcal{NP}$-Hard, and use Lemma 8.3.1 to conclude that $MaxQE$ is $\mathcal{NP}$-Hard to $\frac{1}{2}+\epsilon$-approximate.

Lemma 8.3.2 *For every $\epsilon > 0$, there exists a polynomial reduction from QE to $GapQE_{1,\frac{1}{2}+\epsilon}$*

Proof: Let I be an input for QE. We show a polynomial time algorithm that constructs from I an input I' for $Gap_{1,\frac{1}{2}+\epsilon}$, where $I' \in YesQE_1 \cup NoQE_{\frac{1}{2}+\epsilon}$ (i.e. the output I' satisfies the promise of $GapQE$), and furthermore $I' \in YesQE_1$ if and only if $I \in QE$. (It follows that $I' \in NoQE_{\frac{1}{2}+\epsilon}$ if and only if $I \notin QE$.)

Let us first introduce some notations we use in the proof. Whenever we deal with a sequence I of n quadratic equations with the Boolean variables $x_1, \ldots x_t$, we denote every equation as e_k ($1 \le k \le n$), the coefficients of the quadratic terms on e_k as $c_{i,j}^{(k)}$ and the free term of e_k as $b^{(k)}$. That is, such a sequence is denoted by

$$\left\{ e_k : \sum_{i=1}^{t}\sum_{j=1}^{t} c_{i,j}^{(k)} x_i x_j = b^{(k)} \right\}_{k=1}^{n}$$

Whenever we deal with a sample space $\Omega \subseteq \{0,1\}^n$, we denote its size by m (and use $m = poly(n)$), its l^{th} element as $s^{(l)}$ and the k^{th} bit of $s^{(l)}$ as $s_k^{(l)}$. Thus, $\Omega = \{s^{(1)}, \ldots, s^{(m)}\}$) and $s^{(l)} = (s_1^{(l)}, s_2^{(l)}, \ldots, s_n^{(l)})$.

Let I be an input for QE. The transformation algorithm is the following:

1. Construct an ϵ-bias sample space Ω over $\{0,1\}^n$ of size $m = poly(n)$.
2. Construct I', a QE input with the same variables $x_1, \ldots, x_t$ as I, but with linear combinations of $e_1, \ldots, e_n$ as the quadratic equations. The linear combinations are those corresponding to the elements $s_1, \ldots, s_m$ of Ω. That is, for every l for which $1 \le l \le m$, we define f_l to be the quadratic equation

$$f_l : \sum_{k=1}^{n} s_k^{(l)} e_i$$

which is

$$\sum_{k=1}^{n} s_k^{(l)} \underbrace{\sum_{i=1}^{t}\sum_{j=1}^{t} c_{i,j}^{(k)} x_i x_j}_{LEFT(e_i)} = \sum_{k=1}^{n} s_k^{(l)} \cdot \underbrace{b^{(k)}}_{RIGHT(e_i)}$$

in other words the new l^{th} equation is

$$\sum_{i=1}^{t}\sum_{j=1}^{t}\underbrace{\sum_{k=1}^{n} s_k^{(l)}\cdot c_{i,j}^{(k)}}_{\text{coefficient of } f_l}\cdot x_i x_j = \underbrace{\sum_{k=1}^{n} s_k^{(l)}\cdot b^{(k)}}_{\text{free term of } f_l}$$

We now prove that the this transformation is indeed a polynomial reduction.

Claim 8.3.3 *The transformation can be applied in polynomial time*

Proof: Constructing a polynomial sized ϵ-bias sample space (see Section 8.2) can be done in polynomial time (i.e., polynomial in n and $\frac{1}{\epsilon}$, where here ϵ is a fixed constant). The time required for the second stage (generating the new equations) is linear in the size of the space, i.e. polynomial. ■

Claim 8.3.4 *I is satisfiable $\Longrightarrow I' \in YesQE_1$*

Proof: Suppose that there is an assignment satisfying all the equations of I, and let $\mathcal{T} = (\tau_1, \ldots, \tau_t)$ be this assignment. Notice that an assignment that satisfies some equations, satisfies also every linear combination of them. Consequently $\mathcal{T}$ satisfies all the equations of I', i.e. $I' \in YesQE_1$. ■

Claim 8.3.5 *I is not satisfiable $\Longrightarrow I' \in NoQE_{\frac{1}{2}+\epsilon}$*

Proof: Let I be an unsatisfiable sequence of quadratic equations. We need to prove that no assignment of $x_1, \ldots, x_t$, satisfies a fraction of $\frac{1}{2}+\epsilon$ of the equations of I'. Fixing any assignment $\mathcal{T}$ to the variables of I, we show that $\mathcal{T}$ satisfies less than $\frac{1}{2}+\epsilon$ of the equations in I'. Our argument relies on the following fact:

Fact 8.3.6 *Let $\mathcal{T}$ be an assignment to $x_1, \ldots, x_t$, and let $K \subseteq \{1, \ldots, n\}$ be the set of indices of equations in I that the assignment $\mathcal{T}$* does not *satisfy. Let f be a linear combination of the quadratic equations in I, that is $f = \sum_{k=1}^{n} s_k e_k$ where $s_1, \ldots, s_n \in \{0,1\}$. Then $\mathcal{T}$ satisfies f if and only if*

$$\bigoplus_{k\in K} s_k = 0$$

that is, the number of k's such that $k \in K$ and $s_k = 1$ is even.

Proof: Every s_k that is assigned 1, represents an equation that is included within f. Each of these k's that is in K, represents an equation that is not satisfied by $\mathcal{T}$, and contributes 1 to the difference between $LETF(f)$ and $RIGHT(f)$.

Recall that f is an equation (mod2), and thus a pair of contributions of 1 cancel each other. Thus, whenever the number of these contributions is even, their total difference is summed up to 0, and otherwise a difference of 1 remains, which means that in this case the equation f is not satisfied by the assignment $\mathcal{T}$. ■

Motivating Discussion: Before explicitly proving the upper bound on the number of satisfied equations in I', which has a somewhat technical proof, we first give some intuition for the main ideas in constructing it, via proving such an upper bound for slightly different construction of I'.

Consider the construction of I', which instead of sampling m elements of $\{0,1\}^n$, takes the whole $\{0,1\}^n$ as Ω (note that such an instance I' has exponential size and cannot be constructed in polynomial time from I), which is a perfect sample space. Let $\mathcal{T}$ be an assignment, and recall that I is not satisfiable, and thus the set K of equations that are not satisfied by $\mathcal{T}$ is non-empty.

Recall that the uniform distribution is an 0-bias sample space (as described in Section 8.2), which means that for every non-empty K, the fraction of n-bit vectors, whose $\bigoplus$ over K equals 0, is exactly one half. Thus, we can apply Fact 8.3.6 and conclude that the fraction of s's that induce equations unsatisfied by $\mathcal{T}$, is exactly one half.

We return now to the original construction of I'. Although our sample space has only polynomial size ($m = poly(n)$), recall that the main property of an ϵ-bias sample space is that the parity bit of *every* non-empty subset, has an almost uniform distribution in Ω (it has at most a bias of $\frac{\epsilon}{2}$ towards each of the values in $\{0,1\}$). Consequently, the fraction of linear combinations where the number of unsatisfied e_k's is even, is bounded from above by $\frac{1}{2} + \frac{\epsilon}{2}$ (and from below by $\frac{1}{2} - \frac{\epsilon}{2}$) (details follow).

Let $\mathcal{T}$ be an assignment, and suppose that $\mathcal{T}$ does not satisfy *some* of the equations $e_1, \ldots, e_n$ in I (as must be the case since $I \notin QE$). Let $K \subseteq \{1, \ldots, n\}$ be the set of indices of equations in I that are not satisfied by $\mathcal{T}$. We know that K is not empty as I is not satisfiable. An equation f_l in I' is the linear combination of the equations $\{e_k\}_{k=1}^{n}$, with coefficients corresponding to the element $s^{(l)}$ in Ω. Again we use the observation of Fact 8.3.6 that $\mathcal{T}$ satisfies f_l if and only if $\bigoplus_{k \in K} s_k^{(l)} = 0$. However, the fraction of these $s^{(l)}$'s in Ω is bounded from above by $\frac{1}{2} + \frac{\epsilon}{2}$. This fact stems from the definition of ϵ-bias sample spaces.

Thus, the fraction of f_l's that are satisfied by $\mathcal{T}$ is at most $\frac{1}{2} + \frac{\epsilon}{2}$, which is strictly smaller than $\frac{1}{2} + \epsilon$. ■

Thus, the reduction is polynomial and valid and the lemma follows. ■

Bibliographic Notes

Small-bias sample spaces were introduced and first constructed by Naor and Naor [35]. The construction presented in this lecture is one of three alternative constructions presented by Alon, Goldreich, Håstad and Peralta [8]. The application to hardness of MaxQE is due to Håstad, Phillips and Safra [25], who also demonstrate the tightness of the non-approximability factor (by presenting a suitable polynomial-time approximation algorithm). For an introduction to probabilistic checkable proofs (i.e., PCP), see [21, Sec. 2.4].

Lecture 9

Expanders and Eigenvalues

Notes taken by Ya'ara Goldschmidt, Eran Keydar, Yaki Setty

Summary: This lecture is about expander graphs, which are very useful in many probabilistic applications. First, we would describe some of the combinatorial definitions and properties of an expander graph. Second, we would see one application which uses expander graphs, and finally, we would discuss some of the algebraic properties of such graphs.

9.1 Introduction

9.1.1 Motivation

We have already seen (in Lecture 2) a proof, using the probabilistic method, for the existence of 3-regular expander graphs. In the following lectures we study expanders, their applications and constructions, at greater depth.
As was presented in Lecture 2, expander graphs are sparse graphs with the property that every subset of vertices (which is not too large), has "many" adjacent neighbors. This implies the existence of a "short" path between any two vertices.
These properties are very useful in many probabilistic applications. For example they ensure that when performing a random walk on an expander graph, the distribution of reaching any of the vertices will converge to uniformity, relatively fast.

9.1.2 Family of Expander Graphs

Usually, when speaking on an expander graph G_N, we actually mean a family of graphs $\{G_N\}_{N\in S}$, where S is some infinite subset of the natural numbers. The members of the family are d-regular graphs (d is fixed for the entire family) and have the same expansion coefficient $c > 0$.
Three remarks for the definition of the set of graphs:

1. The graphs are not necessarily simple; that is, self-loops and parallel edges are allowed. That means that the adjacency matrix representing such a graph can have entries in the range 0 to d instead of only boolean entries.

2. In fact, we assume that all vertices have self-loops; that is, the adjacency matrix has 1's on the diagonal.

3. For our purpose, it is convenient to assume that N is equal to 2^n for some n, and that the vertices of the graph are the 2^n strings of length n (i.e., $\{0,1\}^n$).

9.1.3 Definitions of Neighborhood Sets

Before formally defining what an expander graph is, we define some notations about neighborhood sets. For a set of vertices U in graph $G(V,E)$, we define the following three neighborhood sets:

Definition 9.1 $\Gamma(U) \triangleq \{v : \exists u \in U\ ,\ (u,v) \in E\}$*; the set of all vertices reachable from any vertex of* U *in exactly one step (may include vertices in* U*).*

Definition 9.2 $\Gamma^*(U) \triangleq \Gamma(U) \backslash U$*; the set of all vertices reachable in one step from vertices in* U*, excluding* U*. This is called the boundary of* U*.*

Definition 9.3 $\Gamma^+(U) \triangleq \Gamma(U) \cup U$*; the set of all vertices reachable in at most one step from* U*.*

Notice: under the assumption that all vertices have a self-loop, $\Gamma^+(U) = \Gamma(U)$.

9.1.4 Definitions of Expander Graphs

We can now formally define an expander graph.

Definition 9.4 [First Definition] *N-vertex graph has expansion coefficient $c > 0$, if $\forall$ set U of vertices with size $\leq \frac{N}{2}$: $|\Gamma^*(U)| \geq c \cdot |U|$*

Sometimes one would find it more convenient to use the following definition for an expander graph:

Definition 9.5 [Second Definition] *N-vertex graph has expansion $c > 0$, if $\forall$ set U of vertices:* $|\Gamma^+(U)| \geq \min\left\{(1+c)|U|, \frac{N}{2}+1\right\}$

Notice: any graph G satisfying Definition 9.4 also satisfies Definition 9.5, but the converse is not necessarily true.
Definition 9.5 implies that every expander graph is connected, as stated in the following proposition:

Proposition 9.1.1 *Any expander graph is a connected graph.*

Proof: Let $G_N(V,E)$ be an expander graph with expansion coefficient $c > 0$. Suppose in contradiction that G_N is not connected. In that case, V is composed of at least 2 disjoint subsets of vertices, with no edges connecting them. Hence for any such subset U, $\Gamma^+(U) = U$. Let $U_{\min}$ be the subset of smallest size, clearly $|U_{\min}| \leq \frac{N}{2}$, hence $|\Gamma^+(U_{\min})| < \min\left\{(1+c)|U_{\min}|, \frac{N}{2}+1\right\}$. This contradicts the second definition of expander graph (Definition 9.5). ■

9.1.5 Constructibility

When speaking on expander graphs, one should consider the time required to construct the graph. This is the requirement of constructibility. There are two natural versions for this requirement: a weak version and a strong one.

Definition 9.6 [Weak Version] *An expander graph G_N is (weakly) constructible, if an explicit representation of it can be given in time* $\text{poly}(N)$*.*

An explicit representation means an adjacency matrix or a neighbors list.
This definition is weak, since the construction is polynomial in the number of vertices. As mentioned before, in some applications we would like to relate to vertices as strings over $\{0,1\}^n$, and then the output would be exponential in n. This leads to the stronger definition of constructibility:

Definition 9.7 [Strong Version] *An expander graph G_N is (strongly) constructible, if when given a string of length n bits, representing vertex v in G_N ($N = 2^n$), we can construct a list of all its neighbors in time* $\mathrm{poly}(n)$.

Clearly, Definition 9.7 implies Definition 9.6.

9.2 On the Diameter of Expander Graphs

In this section we prove that there is a relatively short path between any two vertices in an expander graph.

Definition 9.8 *The diameter of a connected graph G, denoted $Diam(G)$, is the maximal distance between any two vertices in the graph.*

Theorem 9.9 [Logarithmic Diameter]
Let G_N be an expander graph with an expansion coefficient c, then:

$$Diam(G_N) \leq \frac{2\log(N)}{\log(1+c)}$$

Proof: In order to prove the theorem, we would prove that for every two vertices u and v, there exists a path connecting u and v of length at most $\frac{2\log((N/2)+1)}{\log(1+c)}$.
Define $l = \log_{1+c}(\frac{N}{2}+1)$, and define the following sets inductively:

$$\begin{aligned} U_0 &= \{u\} \\ U_i &= \Gamma^+(U_{i-1}) \end{aligned}$$

Then, U_i is the set of the vertices reachable from u in at most i steps. Using the second definition of expansion (Definition 9.5) we conclude:

$$\begin{aligned} |U_i| &= |\Gamma^+(U_{i-1})| \\ &\geq \min\left\{(1+c)|U_{i-1}|, \tfrac{N}{2}+1\right\} \\ &\geq \min\left\{(1+c)^i, \tfrac{N}{2}+1\right\} \end{aligned}$$

Using $i = l$, we get $|U_l| > \min\left\{(1+c)^l, \frac{N}{2}+1\right\}$, which is at least $\frac{N}{2}+1$ provided $(1+c)^l > \frac{N}{2}$. The latter is correct by the definition of l. Applying the same argument for v we get $|V_l| > \frac{N}{2}$. Remember that U_l and V_l are the sets of vertices reachable in at most l steps from u and v, respectively. Both U_l and V_l are subsets of the set of N vertices, therefore these two sets must have non-empty intersection. Therefore there must be a path between u and v of length at most:

$$2l = 2 \cdot \frac{\log(\frac{N}{2}+1)}{\log(1+c)} \leq \frac{2\log(N)}{\log(1+c)}$$

■

9.3 Amplification of Expander Graphs

For some applications we would need an expansion coefficient that is larger than the one associated with the $\{G_N\}_{N\in S}$ which we are given or have been constructed. To achieve this we amplify the graph G_N, to get a graph with larger expansion coefficient, but also with more edges.

9.3.1 The Amplified Graph

Given the graph G_N with vertex set $[N]$, we can construct the graph G_N^k in the following way:

$$G_N^k \triangleq \left([N], \left\{(u,v) : \begin{array}{c} \exists \text{ path of length exactly } k \\ \text{between } u \text{ and } v \text{ in } G_N \end{array}\right\}\right)$$

Given a representation of G_N^k as an adjacency matrix - M_N, the adjacency matrix M_N^k of G_N^k can be calculated by raising M_N to the k-th power: $M_N^k = (M_N)^k$. We can now present the following proposition about G_N^k constructed from an expander graph G_N which is d-regular and has expansion c.

Proposition 9.3.1 *Let G_N be a d-regular expander graph with expansion coefficient c, and let k be an integer. Then G_N^k has the following properties:*

1. *G_N^k is d^k-regular.*

2. *The expansion coefficient of G_N^k is $(1+c)^k - 1$.*

Proof:

1. By the construction of G_N^k, the degree of each vertex in G_N^k is the number of paths (not necessarily simple) of length k in G_N. Since G_N is d-regular, the number of paths is exactly d^k. Hence G_N^k is d^k-regular.

2. For any set U, we denote by U_i the set of vertices reachable from U in at most i steps in the graph G_N. Then $U_0 = U$, $U_1 = \Gamma^+(U)$, and in general $U_i = \Gamma^+(U_{i-1})$.

$$\begin{aligned} |U_0| &= |U| \\ |U_i| &\geq \min\left\{(1+c)|U_{i-1}|, \tfrac{N}{2}+1\right\} \\ &\geq \min\left\{(1+c)^i|U|, \tfrac{N}{2}+1\right\} \end{aligned}$$

   In G_N^k the set $\Gamma_{G_N^k}(U)$ is the set of all vertices reachable from U in exactly one step, that is, it is equal to the set of vertices reachable in exactly k steps from U in G_N. But since every vertex in G_N has a self-loop, $\Gamma_{G_N^k}(U)$ also equals to the set of vertices reachable from U in at most k steps. This also implies that U is a subset of $\Gamma_{G_N^k}(U)$, hence $\Gamma^+_{G_N^k}(U) = \Gamma_{G_N^k}(U) = U_k$. Therefore $|\Gamma^+_{G_N^k}(U)| \geq \min\left\{(1+c)^k|U|, \frac{N}{2}+1\right\}$. Using the second definition of expansion, the expansion coefficient of G_N^k is $(1+c)^k - 1$.

■

9.3.2 Constructibility of G_N^k

We now present two claims about the constructibility of G_N^k:

Claim 9.3.2 *Let G_N be a d-regular expander graph which is weakly constructibility, and let k be an integer, then G_N^k is weakly constructible as well.*

Proof: We notice that if we take k to be as large as the diameter of the graph, then G_N^k is a complete graph, therefore we can assume that k is not larger than the diameter of the graph. Based on Theorem 9.9, $k = O(\log(N))$. If G_N is weakly constructible then we can construct its adjacency matrix in time poly(N). Raising it to the k-th power, requires at most k times poly(N)-time operations. Since we assumed $k = O(\log(N))$ the whole construction takes poly(N) time. ■

Claim 9.3.3 *Let G_N be a d-regular expander graph which is strongly constructible, and let k be an integer. Then the generation of the list of neighbors of each vertex in G_N^k takes* $\text{poly}(n \cdot |output|)$*; that is,* $\text{poly}(n \cdot d^k)$.

Notice that since k may not be constant, the number of neighbors of a vertex in G_N^k can be exponential in n, therefore G_N^k is not necessarily strongly constructible.
Proof: If G_N is strongly constructible, then the generation of the neighbors list of v in G_N^k can be done by using a BFS search on G_N for depth k. The number of vertices in depth till k is upper bounded by $\sum_{i=1}^{k} d^i$, that is $O(d^k)$. Since finding the neighbors of a vertex in G_N takes poly(n) time, the complexity of the BFS is $\text{poly}(n \cdot d^k)$. And this is the most expensive operation of the construction. Notice that if d^k is polynomial in n, then G_N^k is strongly constructible. ■

9.4 An Application of Expander Graphs

The following section presents one application of expander graphs.

9.4.1 The Problem

Consider the following problem: Suppose W is a set of witnesses, $W \subseteq \{0,1\}^n$ such that $|W| \geq \frac{1}{2} \cdot 2^n$. Our aim is to present a randomized algorithm that when given $\delta < \frac{1}{2}$ (e.g. $\delta = 0.01$) satisfies:

$$\Pr[\text{Output of the Alg} \in W] > 1 - \delta$$

An obvious solution would be to pick $\log\left(\frac{1}{\delta}\right)$ strings independently (each giving a probability of at least $\frac{1}{2}$ to hit W). But we seek for a solution which would satisfy the following restrictions:

1. Running time should be $\text{poly}(\frac{n}{\delta})$

2. At most n bits of randomness are allowed to be used.

The second restriction prevents us from using the above (obvious) solution, and even does not allow to construct a pairwise independent sample space.

9.4.2 The Algorithm

We start with a d-regular expander graph G_N with expansion c, we then construct the graph G_N^l (as explained in Section 9.3), which is d^l-regular and has expansion coefficient $(1+c)^l - 1$. By choosing l to be $\frac{\log(1/\delta)}{\log(1+c)}$, we ensure that the expansion coefficient of G_N^l is approximately $\frac{1}{\delta}$. Using this construction, consider the following algorithm:

1. Select at random a vertex v in G_N^l.

2. Scan the neighbors of v, and output a neighbor in W if such exists, else fail.

Three remarks:

1. Each vertex is represented as a string of length n bits, thus only n bits of randomness are required for Stage 1.

2. We actually do not explicitly construct G_N^k. Instead, after randomly selecting the string v (in Stage 1), we use the fact that G_N is strongly constructible to generate the neighbors of v in G_N^k in time poly$(\frac{n}{\delta})$.

3. Since the graph has self-loops, in Stage 2 of the algorithm v is scanned along with its neighbors.

9.4.3 Analysis of the Algorithm

Time Complexity

Using Claim 9.3.3, the time complexity of Stage 2 of the algorithm is poly$(n \cdot d^l)$, where:

$$d^l = d^{O\left(log\left(\frac{1}{\delta}\right)\right)} = \exp\left(\log\left(\frac{1}{\delta}\right)\right) = \text{poly}\left(\frac{1}{\delta}\right)$$

provided that d is constant. That is, the complexity of Stage 2 is poly$(\frac{n}{\delta})$. This is the most expensive operation of the algorithm, hence this determines its time complexity.

Correctness

We now analyze the probability of the algorithm to fail, which happens when none of the neighbors of the chosen v are in W. We define a vertex as *bad* if all its neighbors (and itself) are in $\{0,1\}^n \backslash W$. Denote $B = \{v : v \text{ is } bad\}$.

Claim 9.4.1 $|B| < \delta \cdot 2^n$

Proof: Suppose in contradiction that $|B| \geq \delta \cdot 2^n (= \delta \cdot N)$. Using the second definition of expansion on $\Gamma^+(B)$ we get:

$$|\Gamma^+(B)| = \min\left\{\underbrace{((1+c)^l - 1)}_{> \frac{1}{\delta}} \cdot \delta N, \frac{N}{2} + 1\right\} = \frac{N}{2} + 1$$

Hence $|\Gamma^+(B)| \geq \frac{N}{2}$, but also $\Gamma^+(B) \cap W = \phi$. This contradicts the hypothesis that $|W| > \frac{N}{2}$. ■

9.5 Algebraic Definition of Expanders

Until now we have been discussing the combinatorial properties of expanders. In the following section we will talk about some algebraic properties of expander graphs. This will lead us, in the next lecture, to the algebraic definition of an expander graph and to the relations between the combinatorial and algebraic definitions. We will study the algebraic properties by exploring the connection they have with a random walk on the graph.
A *random walk* is a sequence of random steps, at which we start from an initial vertex and uniformly select one of its outgoing edges. We follow that edge and repeat the random selection for the vertex we have reached, and so on for any desired number of steps. Since the selection of an outgoing edge in each step is uniform, the probability to reach each vertex in that step is proportional to the number of edges connecting the current vertex to it.

9.5.1 The Normalized Adjacency Matrix

Consider the symmetric adjacency matrix of an undirected d-regular graph with vertex set $[N]$. The entries in the matrix are between 0 and d (since we allow self-loops and parallel edges). We define the *normalized adjacency matrix* to be the above matrix divided by d. This devision ensures that all the entries are in $[0,1]$, and the sum of each column and each row is 1. The *normalized adjacency matrix* is a stochastic matrix, and as such it can be used to describe a random walk on the graph. Denote the *normalized adjacency matrix* by A, then entry a_{ij} is the probability to reach vertex i in one random step, given that we are now in vertex j. We denote this probability by $\Pr[\,i\,|\,j\,]$. We can now look at the following operation:

$$\overbrace{\begin{bmatrix} a_{11} & \dots & a_{1N} \\ \vdots & \ddots & \vdots \\ a_{N1} & \dots & a_{NN} \end{bmatrix}}^{A:N\times N} \underbrace{\begin{bmatrix} p_1 \\ \vdots \\ p_N \end{bmatrix}}_{P:N\times 1} = \underbrace{\begin{bmatrix} p'_1 \\ \vdots \\ p'_N \end{bmatrix}}_{P':N\times 1}$$

We claim that the above operation describes one step of a random walk on the graph: That is, if P is the probability distribution of being in any vertex at the current stage, then P' is the distribution after doing one random step. To explain this claim we look at the i-th entry of P' which is:

$$\begin{array}{rcccccccc} p'_i & = & a_{i1}\cdot p_1 & + & a_{i2}\cdot p_2 & + & \dots & + & a_{iN}\cdot p_N \\ & = & \Pr[\,i\,|\,1\,]\cdot p_1 & + & \Pr[\,i\,|\,2\,]\cdot p_2 & + & \dots & + & \Pr[\,i\,|\,N\,]\cdot p_N \end{array}$$

That is, p'_i describes the probability to reach vertex i after that step, so P' is the probability distribution after that step. Basically, if we start with initial distribution vector $P^{(0)}$, then the distribution after one random step is $P^{(1)} = AP^{(0)}$. After another random step the distribution is $P^{(2)} = AP^{(1)} = A^2P^{(0)}$. In general, the distribution after t random steps would be $P^{(t)} = A^tP^{(0)}$.

We call a distribution P a *stationary distribution* if $AP = P$. Clearly, if for some $t > 0$ it holds that $A^tP^{(0)} = P^{(t)}$ is the stationary distribution, then for any $t' \geq t$, the distribution $P^{(t')}$ remains the stationary distribution. We say that a random walk *converges* if for any initial distribution $P^{(0)}$, and for every $\epsilon > 0$ there exists $t > 0$ such that $A^tP^{(0)} = P^{(t)}$ is ϵ-close (in variation distnace) to the stationary distribution. As a corollary to the facts proven in the next section, one may conclude that for any connected bipartite (regular) graph a random walk on its vertices converges to the uniform distribution. In the next lecture, we shall see that the rate of convergence (i.e., the dependence of t on ϵ) depends on the eigenvalues of the adjacency matrix.

9.5.2 Eigenvectors and Eigenvalues of the Matrix

First, we would explore the eigenvalues and eigenvectors of the (normalized) adjacency matrix:

Fact 9.5.1 *All eigenvalues are real.*

This follows from the fact that the adjacency matrix is symmetric.

Fact 9.5.2 *The eigenvectors form a basis.*

This again follows from the fact that the adjacency matrix is symmetric and therefore diagonalizable.

Fact 9.5.3 *The value* 1 *is an eigenvalue corresponding to the uniform eigenvector* $\bar{e}_1 = (\frac{1}{N}, \ldots, \frac{1}{N})$.

That is: $A \cdot \bar{e}_1 = \bar{e}_1$.
Proof: Indeed examine $\bar{p} = A \cdot \bar{e}_1$. For any entry $1 \le i \le N$ in $\bar{p}$:

$$p_i = \sum_{j=1}^{N} a_{ij} \frac{1}{N} = \frac{1}{N} \sum_{j=1}^{N} a_{ij} = \frac{1}{N} \cdot 1 = \frac{1}{N}$$

■

Remark : $\bar{e}_1$ is the *stationary distribution.*

Fact 9.5.4 *Eigenvectors corresponding to distinct eigenvalues are orthogonal. That is, if* $A\bar{v_1} = \lambda_1 \bar{v_1}$, $A\bar{v_2} = \lambda_2 \bar{v_2}$ *and* $\lambda_1 \neq \lambda_2$, *then* $\bar{v_1}^\top \bar{v_2} = 0$ *(i.e.* $\bar{v_1} \perp \bar{v_2}$*).*

Proof: Examine $\bar{v_1}^\top A \bar{v_2}$
On the one hand:

$$\bar{v_1}^\top A \bar{v_2} = (A^\top \bar{v_1})^\top \bar{v_2} \overset{(*)}{=} (A\bar{v_1})^\top \bar{v_2} = (\lambda_1 \bar{v_1})^\top \bar{v_2} = \lambda_1 \bar{v_1}^\top \bar{v_2}$$

where $(*)$ holds since $A^\top = A$.
On the other hand:

$$\bar{v_1}^\top A \bar{v_2} = \bar{v_1}^\top \lambda_2 \bar{v_2} = \lambda_2 \bar{v_1}^\top \bar{v_2}$$

Hence $\lambda_1 \bar{v_1}^\top \bar{v_2} = \lambda_2 \bar{v_1}^\top \bar{v_2}$. Thus $\lambda_1 \neq \lambda_2$ implies $\bar{v_1}^\top \bar{v_2} = 0$. ■

Fact 9.5.5 *Any eigenvector* $\bar{v}$ *corresponding to an eigenvalue* $\neq 1$ *has sum* 0.

Proof: This is deduced from Fact 9.5.4. Since $\bar{v}$ is orthogonal to the uniform vector $\bar{v}^\top \cdot \begin{bmatrix} \frac{1}{N} \\ \vdots \\ \frac{1}{N} \end{bmatrix}$ $= 0$.
This implies that $\sum_{i=1}^{N} v_i \frac{1}{N} = 0$, therefore $\frac{1}{N} \sum_{i=1}^{N} v_i = 0$, and so $\sum_{i=1}^{N} v_i = 0$. ■

Fact 9.5.6 *For any eigenvalue* λ, $|\lambda| \le 1$.

Proof: Let $\{\bar{e}_i\}$ be the eigenvectors, where $\bar{e}_1 = (\frac{1}{N}, \ldots, \frac{1}{N})$.
Any probability vector $\bar{p}$ can be represented as a linear combination of the basis $\{e_i\}$ (Fact 9.5.2):

$\bar{p} = \sum_{i=1}^{N} c_i \bar{e}_i$.
Let $t \geq 1$ be an integer.

$$A^t \bar{p} = A^t \sum_{i=1}^{N} c_i \bar{e}_i = \sum_{i=1}^{N} c_i A^t \bar{e}_i = \sum_{i=1}^{N} c_i {\lambda_i}^t \bar{e}_i$$

If $\lambda_i > 1$ and $c_i \neq 0$ then the above sum diverges, which is not possible since $A^t \bar{p}$ is a probability vector.
However if $c_i = 0$, then we cannot conclude that $|\lambda_i| \leq 1$. Thus, for any $i \geq 2$, consider $\bar{p} = \bar{e}_1 + \epsilon \bar{e}_i$, where ϵ is small enough such that $\bar{p}$ is a probability vector. That is:

$$\epsilon \leq \frac{1}{|m|N} \qquad \text{where } m = \min_{1 \leq j \leq N} \{e_{i_j}\}$$

Taking such ϵ ensures that $\bar{p}$ is a probability vector:

- All entries of $\bar{p}$ are non-negative:
  Since m is the minimal entry of $\bar{e}_i$, all entries in $\epsilon \bar{e}_i$ are greater or equal to $-\frac{1}{N}$. Moreover, all entries in $\bar{e}_1$ are $\frac{1}{N}$, therefore all the entries in $\bar{p}$ are non-negative.
- The sum of the entries of $\bar{p}$ is 1:
  The sum of entries in $\bar{e}_1$ is 1, and the sum of entries in $\bar{e}_i$ is 0 (Fact 9.5.5), hence the sum of entries in $\bar{e}_1 + \epsilon \bar{e}_i$ is 1.

Now consider:
$$\bar{p}' = A^t \bar{p} = A^t(\bar{e}_1 + \epsilon \bar{e}_i) = A^t \bar{e}_1 + A^t \epsilon \bar{e}_i = \bar{e}_1 + \epsilon {\lambda_i}^t \bar{e}_i$$

$\bar{p}'$ should also be a probability vector, but when $|\lambda_i| > 1$ and for t large enough, the above term diverges. Hence, we conclude that $|\lambda_i| \leq 1$ ■

Fact 9.5.7 *If G is connected, then the eigenvalue 1 has a unique eigenvector.*

Proof: Suppose in contradiction, that $\bar{e}_1$ and $\bar{e}_2$ are two orthogonal eigenvectors corresponding to the eigenvalue 1. Let $\bar{p} = \bar{e}_1 + \epsilon \bar{e}_2$, where $\epsilon = \frac{1}{|m|N}$ and m is the minimal entry of $\bar{e}_2$. Such ϵ ensures that for any entry with value m in e_2, the corresponding entry in $\bar{p}$ is 0 (see the proof of Fact 9.5.6). That means that $\bar{p}$ has some zero entries and the rest are positive entries. Since $\lambda_1 = \lambda_2 = 1$, the following holds:

$$A\bar{p} = A(\bar{e}_1 + \epsilon \bar{e}_2) = A\bar{e}_1 + \epsilon A \bar{e}_2 = 1 \cdot \bar{e}_1 + \epsilon \cdot 1 \cdot \bar{e}_2 = \bar{e}_1 + \epsilon \bar{e}_2 = \bar{p}$$

Therefore for any positive t, $A^t \bar{p} = \bar{p}$. Let i be an index of a positive entry, and let j be an index of a zero entry. Thus, the i-th entry in $\bar{p}$ is positive, whereas the j-th entry in $A^t \bar{p}$ is zero. It follows that, for any integer $t > 0$, the probability to reach from vertex i to vertex j in t steps is 0 (since $(A^t \bar{p})_j$ is the sum of the probability of reaching j in t steps, when starting in vertex k with probability p_k). This implies that for any $t > 0$, there is no path of length t between i and j which implies that the graph is not connected, thus we get contradiction. ■

Fact 9.5.8 *For a connected aperiodic graph G, -1 is not an eigenvalue.*

Definition 9.10 *A graph G is said to be aperiodic if the GCD of its cycle* (not necessarily simple cycles) *lengths is* 1.

We first notice that the GCD of all cycle lengths is at most 2. If the GCD is 2, then the graph must be bipartite with no self loops. In this case all cycles would be of even length. However, since the graph is aperiodic its GCD is 1, hence there must exist an odd cycle on the graph.

Proof: Suppose in contradiction that $\bar{e}_2$ is an eigenvector with a corresponding eigenvalue -1. Let $\bar{p} = \bar{e}_1 + \epsilon\bar{e}_2$, where $\epsilon = \frac{1}{|m|N}$ (as in the proof of Fact 9.5.7). Let u be an index of a zero entry in $\bar{p}$, hence negative in $\bar{e}_2$. First we claim that since there exists an odd cycle in the graph, then there is an odd cycle going through u. Such cycle can be constructed by combining the path from u to the odd cycle, the cycle itself, and the path back to u. (The path from u to the odd cycle exists because the graph is connected.) Let l be the length of such an odd cycle. Consider $A^{l+1}\bar{p}$: On the one hand:

$$A^{l+1}\bar{p} = A^{l+1}(\bar{e}_1 + \epsilon\bar{e}_2) = A^{l+1}\bar{e}_1 + A^{l+1}\epsilon\bar{e}_2 = {\lambda_1}^{l+1}\bar{e}_1 + \epsilon{\lambda_2}^{l+1}\bar{e}_2 = \bar{e}_1 + (-1)^{l+1}\epsilon\bar{e}_2 = \bar{p}.$$

In particular, the u-th entry of $A^{l+1}\bar{p}$ is 0 (just as this entry is in $\bar{p}$).
On the other hand:

$$A^{l+1}\bar{p} = A^l(A\bar{p}) = A^l(\lambda_1\bar{e}_1 + \epsilon\lambda_2\bar{e}_2) = A^l(\bar{e}_1 - \epsilon\bar{e}_2).$$

Since the u-th entry of $\bar{e}_2$ is negative, the u-th entry of $(\bar{e}_1 - \epsilon\bar{e}_2)$ is positive. Since u is a vertex on an odd cycle with length l, there must be a positive probability to reach u in l steps when starting from u. Hence the u-th entry of $A^l(\bar{e}_1 - \epsilon\bar{e}_2) = A^{l+1}\bar{p}$ must be positive. This contradicts the previous conclusion that the u-th entry equals 0. ■

In particular, this fact holds for expanders, since the existence of self-loops implies they are aperiodic.

Bibliographic Notes

The elementary facts regarding the eigenvalues of a symmetric matrix can be found in many standard textbooks on linear algebra. The properties of eigenvalues and eivenvectors of matrices associated with graphs were investigated in the combinatorics literature; see [9] and the references therein.

The relation of the algebraic definition (i.e., the second eigenvalue of the normalized adjacency matrix of a graph) and the combinatorial definition of expansion is further discussed in the next lecture (and the most relevant references are [9, 5]).

Lecture 10

Random Walks on Expanders

Notes taken by Omer Angel, Dani Halevi, Amos Gilboa

Summary: The purpose of this lecture is to show that taking an l step random walk in an expander graph is in a way similar to choosing l vertices at random. The advantage of using a random walk on an expander over uniform independent sampling is that it requires only $\log_2 d$ random bits for each step, whereas when you choose l different vertices at random each vertex requires $\log_2 N$ random bits. As a warm up, we'll show that in $O(\log N)$ random steps on an N-vertex expander beginning at any vertex we'll reach a roughly uniform distribution on the vertices.

We also recall the relation between two definitions of expanders and describe some known constructions of expanders.

10.1 Two Definitions

Recall the combinatorial and algebraic definitions of expansion:

Definition 10.1 *A sequence of graphs $\{G_N\}$ has combinatorial expansion C if any of vertices S with $|S| < |V|/2$ has $|\Gamma^{-}(S)| > C|S|$ where $\Gamma^{-}(S)$ denotes the neighbours of a set S that are not in S.*

Definition 10.2 *A sequence of graphs $\{G_N\}$ has algebraic expansion with gap $\epsilon > 0$ if for all n the second eigenvalue of G_n satisfies $|\lambda_2| \leq 1 - \epsilon$.*

remark. We assume that the eigenvalues are ordered by non-increasing order, i.e.

$$1 = \lambda_1 > |\lambda_2| \geq \ldots |\lambda_n|$$

(Recall that we proved that for the normalized adjacency matrix of any connected regular and non-bipartite graph $\lambda_1 = 1$ and $|\lambda_i| < 1$ for all $i > 1$.)

The following theorem implies that the combinatorial definition of expansion based on boundary sizes and the algebraic definition based on eigenvalue separation are equivalent. In fact it gives a quantitative translation between the two notions. (This Theorem is not proved here).

Theorem 10.3 *Algebraic expansion $\Longleftrightarrow$ Combinatorial expansion. In particular:*

- *If $\{G_N\}$ are expanders w.r.t. the algebraic definition with constant ϵ then they are expanders w.r.t. the combinatorial definition with constant $C \geq \frac{2\epsilon}{1+2\epsilon}$.*
- *If $\{G_N\}$ are d-regular expanders w.r.t. the combinatorially definition with constant C then they are expanders w.r.t the algebraic definition with $\epsilon \geq \frac{C^2}{(4+2C^2)d}$.*

Note. These inequalities are not tight. One can see it by beginning with expansion parameter C, substituting in the second inequality to derive ϵ, and substituting this ϵ in the first inequality to derive

$$C' \geq \frac{C^2}{C^2 + (2 + C^2)d}$$

Since this way we got $C' < C$, it follows that at least one of the inequalities is not always tight.

Using the second inequality, we have a practical and easy way to conclude that a graph does not have an expansion C, just calculate ϵ and check whether $\epsilon \geq \frac{C^2}{(4+2C^2)d}$, if this is not the case - obviously the graph does not have expansion C. On the other hand if $\epsilon \leq \frac{C}{2-2C}$ then the graph certainly has expansion C. However, as we just observed the inequalities are not tight, so if we find that ϵ is in the interval $[\frac{C^2}{(4+2C^2)d}, \frac{C}{2-2C}]$ then we can not determine from the theorem whether or not the graph has expansion C. Thus this theorem is more important as a qualitative statement.

Just as there is a relation between the combinatorial expansion of a graph G and the expansion of its power G^k (i.e. given by $1 + C_{G^k} > (1 + C_G)^k$) there is also a similar relation between the algebraic expansions of G and G^k. This is given by:

Proposition 10.1.1 $\lambda_2(G^k) = \lambda_2(G)^k$

Proof: The normalized adjacency matrix of G^k is given by $A' = A^k$ (i.e. A raised to power k). If v is an eigenvector of A with eigenvalue λ then

$$A'v = A^k v = \lambda^k v$$

So the eigenvalues of G' are the k'th powers of the eigenvalues of G. Since A has a basis of eigenvectors there are no addition eigenvalues of A'. ■

10.2 Known Constructions of Expanders

Following are two known ways to build big expander graphs. These graphs are strongly constructible (i.e. given a vertex, we can find it's neighbours in time poly(n), where n is the number of bits needed to represent a vertex).

- For any N such that $N = m^2$ for some integer m, we construct an expander in the following way: the set of vertices will be the set of all pairs (x, y) with $x, y \in \{0, \ldots, m-1\}$. The neighbours of a vertex (x, y) are the pairs

  $$\{(x, y \pm (2x + \sigma)) : \sigma \in \{0, 1\}\} \cup \{(x \pm (2y + \sigma), y) : \sigma \in \{0, 1\}\}$$

  where operations are modulo m. Each vertex can be represented in $2 \log m$ bits its neighbours can be found in time poly($\log m$).

  This graph is 8-regular and has $\lambda \approx 0.9$.

- For primes p, q primes such that $p, q \equiv 1 \pmod 4$ and $(\frac{p}{q}) = 1$ and for any integer k there is an N-vertex expander with $N = \frac{q^k(q^{2k}-1)}{2}$. The vertices are 2×2 matrices over the field with q^k elements ($GF(q^k)$). This graph is $(p+1)$-regular and has $\lambda \approx \frac{2}{\sqrt{d}}$.

  These graphs are less convenient to work with, because of the constraints on p, q (and so on N). However, it is still easy to find the neighbours of any given vertex. They also have the advantage of having λ close to optimal, as seen from:

Theorem 10.4 *For any d-regular graph $\lambda_2 \geq \frac{2}{\sqrt{d}}$*

10.3 Random Walks on Expanders

Throughout this section λ will denote an upper bound on $|\lambda_2|$.

10.3.1 Mixing Time of Random Walks

Let $X_0, X_1, \ldots, X_l$ be a random walk on an expander, starting at X_0 (i.e. X_0 is given either deterministically or by some distribution, and X_{i+1} is chosen uniformly at random from the neighbours of X_i). We show that for $l = O(\log N)$ the distribution of X_l tends to the uniform distribution on V, regardless of the starting point (or its distribution). In a way this number of steps is optimal. This is due to the fact that the diameter of an expander graph is logarithmic in N (because the graph has constant degree). Thus we can't expect to get from one vertex to some other vertices in less than $O(\log n)$ steps.

Theorem 10.5 *Let $X_0, X_1, \ldots, X_l$ be a random walk on an expander graph starting at X_0. For every δ there is an $l = O(\log \frac{1}{\delta})$ such that for every vertex v*

$$\left| \Pr(X_l = v) - \frac{1}{N} \right| < \delta$$

Note. The important fact here is not the dependence on δ which is the same for every graph (not just expanders). The difference between expanders and other graphs is that the constant implied by the $O()$ notation does not depend on the size of the graph. As the proof will show, the dependence of l on the second eigenvalue λ_2 is $l = \frac{\log 1/\delta}{\log 1/\lambda_2}$. If the bound on the eigenvalues $\lambda = 1 - \epsilon$ is close to 1 then we get

$$l \approx \frac{\log 1/\delta}{\log 1/(1-\epsilon)} \approx \frac{\log 1/\delta}{\epsilon}$$

For a fixed δ, the same l will work for all graphs with a given bound λ. Typically we will want to use $\delta \leq 1/2N$. For that value of δ we have

$$\frac{1}{2N} < \Pr(X_l = v) < \frac{3}{2N}$$

So that the probability of being at any vertex is within a constant factor of the uniform distribution. For $\delta = 1/2N$, we need to have $l = O(\log N)$, so we get:

Corollary 10.6 *If $\{G_N\}$ are expander graphs then after $l = O(\log N)$ random steps we are within constant factor of the uniform distribution.*

Proof: [Proof of Theorem 10.5] Let X be a random vertex in G with probability vector v (i.e. v_i is the probability that $X = i$). Let Y be a uniformly chosen neighbour of X. We claim that the probability vector of Y is given by Av where A is the normalized adjacency matrix of G. To see this write the Bayesian equation:

$$\begin{aligned}\Pr(Y = i) &= \sum_j \Pr(Y = i|X = j)\Pr(X = j) \\ &= \sum_j A_{ij} v_j \\ &= (Av)_i\end{aligned}$$

Applying this recursively we see that if the distribution of X_0 is given by a probability vector v then the distribution of X_l is given by $A^l v$.

Next we write v as the sum of the uniform probability vector $\pi = (N^{-1}, \ldots, N^{-1})^T$ and another vector $v_\perp = v - \pi$. Since the sum of the coordinates in v is 1 and the same is true of π we see that the sum of the coordinates of $v_\perp$ is 0. This shows that $v_\perp$ is perpendicular to π justifying the notation $v_\perp$.

A is a real symmetric matrix and as such has perpendicular eigenvectors. Thus $v_\perp$ is spanned by the remaining eigenvectors of A (other then π). We can now estimate the difference in probabilities:

$$\begin{aligned}\|A^l v - \pi\|_\infty &\leq \|A^l v - \pi\|_2 \\ &= \|A^l(v_\perp + \pi) - \pi\|_2 \\ &= \|A^l v_\perp + A^l \pi - \pi\|_2 \\ &= \|A^l v_\perp\|_2\end{aligned}$$

Here we used the fact that π is an eigenvector of A with eigenvalue 1. Since all eigenvalues of A except 1 are bounded by λ the norm of A as an operator on the space perpendicular to π is $|\lambda_2|$. Since $v_\perp$ is perpendicular to π we have:

$$\begin{aligned}\|A^l v_\perp\|_2 &\leq \lambda^l \|v_\perp\|_2 \\ &\leq \lambda^l \|v\|_2 \\ &\leq \lambda^l \|v\|_1 \\ &= \lambda^l\end{aligned}$$

since $v_\perp$ is the projection of v on the space perpendicular to π it has smaller $\|\cdot\|_2$ norm then v and v is a probability vector.

If $\lambda^l < \delta$ then we conclude that for every vertex $|\Pr(X_l = v) - N^{-1}| < \delta$. This happens whenever

$$l > \frac{\log 1/\delta}{\log 1/\lambda} = O(\log \frac{1}{\delta})$$

■

It is interesting to note the combinatorial approach to estimating the mixing time of a random walk. The combinatorial definition of an expander says that there are no small cuts in the graph. Specifically it gives a relation between the size of a set and the number of edges leaving it. (The requirement that $|S| < |V|/2$ is not important since if S contains half the vertices we can remove one from S and find that S still has many neighbours.) There are estimates of the mixing time of the random walk on a graph in terms of the sizes of cuts in the graph and vice versa, but they will not be shown here. We will note, however, that absence of small cuts is a necessary requirement for fast mixing. If a graph is composed of two large vertex sets with a small number of edges (e.g., a single edge) between them then the random walk on the graph started in one set will take a long time to reach the other set.

10.3.2 A random walk yields a sequence of "good samples"

Consider the following problem: In a large set V there is a large subset G of good elements (satisfying some condition) and we wish to find one of them. By large we will mean that the size of G is a constant fraction of the size of V. Call the set of bad elements B and denote its density by $\rho(B) = |B|/|V|$. The set G (and B) are given by an oracle allowing us to check whether some x is in G or in B.

Deterministic algorithms have the drawback that they may take very long for some particularly bad sets B (It is possible to see most of V before finding a point in G). Some randomness must be used to avoid the possibility of terrible sets (at the cost of introducing a probability of error). The trivial random algorithm is to randomly choose elements in V and checking if they are in G. If we make l samples independently then the probability of not finding one in G is $\rho(B)^l$.

This is close to the optimal success probability. The optimal success probability is achieved by also ensuring that the samples are distinct, but since $l \ll |V|$ the improvement is negligible. The drawback here is that this requires the use of many ($\log|V|$) random bits for each sample used.

Expanders can be used to improve this. The algorithm we consider is as follows: Using an expander $G = (V, E)$, Generate l samples in V by taking $X_1 \in_R V$ uniformly at random and $X_2, \ldots, X_l$ are a random walk on G started at X_1. Use the X_i's as samples, while checking if $X_i \in B$. The algorithm is successful if not all X_i fall in B.

Since at each step we only need to choose randomly one of d neighbours (if G is d-regular) then the number of random bits needed is $\log|V| + (l-1)\log d$. On the other hand we will see that the probability of success is comparable to that of the trivial random algorithm (with the same number of vertex checked).

Theorem 10.7 *Let $G = (V, E)$ is d-regular with $|\lambda_2| = \lambda < 1$ and $B \subset V$ has density $\rho = \frac{|B|}{|V|}$. Choose $X_1 \in_R V$ uniformly at random and let $X_2, \ldots, X_l$ be a random walk on G starting at X_1.*

$$\Pr(\forall i\, X_i \in B) \leq \sqrt{\rho} \cdot (\sqrt{\rho} + \lambda)^{l-1}$$

Note this estimate is not quite the same as for independent samples. First of all, even if λ is very small the bound is only $\rho(B)^{l/2}$ compared to $\rho(B)^l$ for independent samples. While this is indeed a difference, the most harm it can do is force us to use twice as many random steps as independent samples, hence the damage is at most multiplication by a constant of complexity and random bits needed. In addition, it is possible to prove the stronger estimate

$$\Pr(\forall i\, X_i \in B) \leq \rho(B) \cdot (\rho(B) + \lambda(1 - \rho(B))^{l-1}$$

which when λ is close to 0 is close to $\rho(B)^l$.

If λ is not very small then the bound is even worse. This drawback can be alleviated by reducing λ by using instead of G the power G^k. By taking k large enough we can make λ smaller. However, we don't gain much by doing this since each random step on G^k takes as many random bits as k

steps on G. Using G^k is equivalent to doing k times as many steps and only checking if $X_i \in Good$ on every k'th step. Since we are taking these steps anyway, usually there is no loss in checking the intermediate vertices as well, i.e. using the random walk on G and not G^k.

One situation where using G^k rather then G may be useful is when the cost of checking whether or not some X is good or bad is not negligible. Then it may be better to take a few more random steps between checks to increase the independence of the positions checked.

Proof: Let $P = P_B$ be a projection on the space of vectors supported in B, i.e. P is a matrix with $P_{ij} = 1$ if $i = j \in B$ and 0 otherwise. If v is a distribution vector then Pv is the residual distribution vector of the distribution v conditioned on being in the set B.

We claim that the probability we wish to estimate is given by: $\|(PA)^{l-1}P\pi\|_1$ where as before, $\pi = (1/N, \dots, 1/N)^T$ is the uniform distribution. To see this, we need to understand the actions of A and P on probability vectors. The action of A is known: for a probability vector v, Av is the probability vector after a random step has been taken.

The action of P is to nullify all the coordinates outside the set B. This transforms a probability vector into the residual probability vector of the same distribution, but conditioned on being in B. Thus $P\pi$ is the residual probability of the uniform distribution conditioned to be in B. $AP\pi$ is the residual distribution after a random step has been takes. $PAP\pi$ is the residual probability conditioned on the random step remaining in B. Repeating this we see that $(PA)^{l-1}P\pi$ is the residual probability vector of the random initial point and all $l-1$ steps being in B. Since we don't care where in B we end up we need to sum the coordinates of this vector, hence the required probability is indeed given by $\|(PA)^{l-1}P\pi\|_1$.

To estimate this we will use the following lemma:

Lemma 10.3.1 *For any non negative vector v:*

$$\|PAv\|_2 \leq (\sqrt{\rho} + \lambda) \cdot \|v\|_2$$

Given this lemma we proceed as follows: using the bound on the ratio between $\|v\|_1$ and $\|v\|_2$ and applying the lemma l times:

$$\begin{aligned} \|(PA)^{l-1}P\pi\|_1 &\leq \sqrt{N} \cdot \|(PA)^{l-1}P\pi\|_2 \\ &\leq \sqrt{N} \cdot (\sqrt{\rho} + \lambda)^{l-1}\|P\pi\|_2 \\ &= \sqrt{N} \cdot (\sqrt{\rho} + \lambda)^{l-1}\sqrt{\frac{\rho}{N}} \\ &= \sqrt{\rho} \cdot (\sqrt{\rho} + \lambda)^{l-1} \end{aligned}$$

Since $\|P\pi\|_2 = \sqrt{|B|/N^2} = \sqrt{\rho/N}$.

■

Proof: [Proof of Lemma 10.3.1] P. The idea of the proof is that A shrinks all components of a vector except the uniform distribution component, whereas P will shrinks the uniform component without increasing anything else. Together they reduce all parts of a vector.

To prove the above formally, break up v as before: $v = v_1 + v_\perp$ where v_1 is a constant vector, i.e. a multiple of the uniform distribution vector, and $v_\perp$ is orthogonal to the uniform distribution vector. In the rest of the proof all norms are Euclidean.

$$\|PAv\| = \|PAv_1 + PAv_\perp\| \leq \|PAv_1\| + \|PAv_\perp\|$$

Next we note how PA effects each part of v. Assume $v_1 = (a, a, \ldots, a)^T$. Then we have $\|v_1\| = \sqrt{N}a$. Since v_1 is an eigenvector of A with eigenvalue 1:

$$\|PAv_1\| = \|Pv_1\| = \sqrt{|B| \cdot a^2} = \sqrt{\rho N} \cdot a = \sqrt{\rho} \cdot \|v_1\|$$

Next we look at the effect on $v_\perp$. P's effect is to multiply some coordinates by 0 without changing the others, so P can only shrink a vector. Since $v_\perp$ is perpendicular to the constant vector, it is spanned by the remaining eigenvectors of A, all with eigenvalues at most λ. This implies that

$$\|PAv_\perp\| \leq \|Av_\perp\| \leq \lambda \|v_\perp\|$$

Combining the two inequalities together with

$$||v_1||, ||v_\perp|| \leq ||v||$$

We see that

$$\|PAv\| \leq \|PAv_1\| + \|PAv_\perp\| \leq \sqrt{\rho}\|v_1\| + \lambda\|v_\perp\| \leq (\sqrt{\rho} + \lambda)\|v\|$$

as required. ■

Bibliographic Notes

Theorem 10.3 (relating the second eigenvalue and the combinatorial definition of expansion) is taken from Alon's work (see [5, Thm. 2.5]): The lower bound on expansion in terms of the second eigenvalue is stated in [5, Cor. 2.3], which in turn relies on [9, Thm. 2.5]. The converse direction is stated in [5, Lem. 2.4].

The two explicite construction of expanders are due to Gaber and Galil [18] (following Margulis [33]) and Lubotzky, Phillips and Sarnak [31], respectively. Actually, the latter construction was originally presented (in [31]) only for primes (i.e., $k = 1$), and the extension to prime powers in due to Alon et. al. [7].

The hitting property of random walks on expanders was discovered by Ajtai, Komlos, and Szemerédi [3]. Various formulations of this discovery were given in [3, 17, 27, 22] culminating in Kahale's optimal analysis [30, Sec. 6]. (A generalization to the case of hitting different sets is presented in [11].)

The fact that a random walk on an arbitrary regular graph reaches almost unifom distribution after polynomially many steps was established in [4].

Appendix

Oded's Note: *Adapted from an early version of [23].*

We complement the material in this lecture by presenting two lemmas related to the properties of one random step taken on a d-regular expander.

Recall that when we talk of expanders we refer to families of d-regular graphs where a fixed upper bound holds for the absolute value of all eigenvalues (except the biggest one) of the corresponding adjacency matrix. Unlike the presentation in the lecture, here it is more convinient to consider the adjacency matrix itself (rather than a normalized (by division with d) form of it). Thus, the upper bound on the second eigen-value, denoted λ, is smaller than d (rather than being smaller than 1). (Similarly, the first eigenvalue is d (rather than being 1).)

The Expander Mixing Lemma

The following lemma is folklore and has appeared in many papers (cf. Corollary 2.5 in [10, Chap. 9]). Loosely speaking, the lemma asserts that expander graphs (for which $d \gg \lambda$) have the property that the fraction of edges between two large sets of vertices approximately equals the product of the densities of these sets. This property is called *mixing*.

Expander Mixing Lemma: *Let $G = (V, E)$ be an expander graph of degree d and λ be an upper bound on the absolute value of all eigenvalues, except the biggest one, of the adjacency matrix of the graph. Then for every two subsets, $A, B \subseteq V$, it holds*

$$\left|\frac{|(A\times B)\cap E|}{|E|} - \frac{|A|}{|V|}\cdot\frac{|B|}{|V|}\right| \leq \frac{\lambda\sqrt{|A|\cdot|B|}}{d\cdot|V|} < \frac{\lambda}{d}$$

Proof: Let $A, B \subseteq V$ be two sets and denote $N \stackrel{\text{def}}{=} |V|$, $\rho(A) \stackrel{\text{def}}{=} |A|/N$ and $\rho(B) \stackrel{\text{def}}{=} |B|/N$. Denote by M the adjacency matrix of the graph G, and let us denote it eigenvalues by $\lambda_1, ..., \lambda_N$, where $|\lambda_i| \geq |\lambda_{i+1}|$. Note that $\lambda_1 = d$, whereas, by the statement of the lemma, $\lambda \geq |\lambda_2|$. Hence, the claim of the lemma is restated as

$$\left|\frac{|(A\times B)\cap E|}{d\cdot N} - \rho(A)\cdot\rho(B)\right| \leq \frac{\lambda\sqrt{\rho(A)\cdot\rho(B)}}{d}$$

We proceed by bounding the value of $|(A \times B) \cap E|$ (from both directions). To this end we let $\bar{a}$ denote the N-dimensional Boolean vector having 1 in the i^{th} component iff $i \in A$. The vector $\bar{b}$ is defined similarly. Clearly, $|(A \times B) \cap E|$ equals $\bar{a}M\bar{b}^\top$. We consider the orthogonal eigenvector basis, $\bar{e}_1, ..., \bar{e}_N$, where $\bar{e}_i\bar{e}_i^\top = N$ for each i, and write each vector as a linear combination of the vectors in the basis, denoting by a_i the coefficient of $\bar{a}$ in the direction of $\bar{e}_i$ (i.e., $\bar{a} = \sum_i a_i\bar{e}_i$). Recall that $\bar{e}_1$ is the constant vector (i.e., with 1's in all entries), and all other $\bar{e}_i$'s are zero-sum vectors. One can easily verify the following two facts:

1. $a_1 = \rho(A)$.

   This follows because the entries in $\bar{a}$ sum-up to $|A|$, and so this must be the sum of the entries in $a_1\bar{e}_1$ (since all other $\bar{e}_i$'s are zero-sum vectors).

2. $\sum_{i=1}^N a_i^2 = \rho(A)$.

   This follows because, on one hand $\bar{a}\bar{a}^\top = \rho(A)N$, and on the other hand $\bar{a}\bar{a}^\top = \sum_{i,j} a_ia_j\bar{e}_i\bar{e}_j^\top = N \cdot \sum_{i=1}^N a_i^2$.

Similarly for $\bar{b}$ (i.e., $\bar{b} = \sum_i b_ie_i$). It now follows that

$$\begin{aligned}
|(A\times B)\cap E| &= \bar{a}M\bar{b}^\top \\
&= \bar{a}M(b_1\bar{e}_1^\top + \sum_{i=2}^N b_i\bar{e}_i^\top) \\
&= \rho(B)\bar{a}M\bar{e}_1^\top + \sum_{i=2}^N b_i\bar{a}M\bar{e}_i^\top \\
&= \rho(B)d\bar{a}\bar{e}_1^\top + \sum_{i=2}^N b_i\lambda_i\bar{e}_i^\top
\end{aligned}$$

$$
\begin{aligned}
&= d\cdot\rho(B)\cdot|A| + \left(\sum_{j=1}^{N} a_j\bar{e}_j\right)\left(\sum_{i=2}^{N}\lambda_i b_i e_i^\top\right)\\
&= \rho(B)\rho(A)\cdot dN + \sum_{i=2}^{N}\lambda_i a_i b_i N\\
&\in \left[\rho(B)\rho(A)\cdot dN \pm \lambda\cdot N\sum_{i=2}^{N} a_i b_i\right]
\end{aligned}
$$

Using $\sum_{i=1}^{N} a_i^2 = \rho(A)$ and $\sum_{i=1}^{N} b_i^2 = \rho(B)$, and applying Cauchy-Schwartz Inequality, we bound $\sum_{i=2}^{N} a_i b_i$ by $\sqrt{\rho(A)\rho(B)}$. The lemma follows. ■

The Expander Smoothing Lemma

The following lemma follows easily by the standard techniques of dealing with random walks on expander graphs (cf. [3] and [10, Chap. 9]). Loosely speaking, the lemma refers to a process that starts with a distribution on the expander vertices and consists of taking one random step on the expander. The lemma bounds the variation distance of the end distribution from the uniform one, in terms of the initial distribution and the quality of the expander. Typically, the end distribution is much closer to the uniform distribution than the initial distribution. That is, a random step on the expander "smoothens" the distribution.

Expander Smoothing Lemma: *Let $G=(V,E)$, d and λ be as in the previous lemma. Let X be a random variable, distributed over V, so that $\Pr(X=v)\leq\frac{K}{|V|}$, for every $v\in V$, and Y denote the vertex reached from X by following a uniformly chosen edge. Then*

$$\sum_{v\in V}\left|\Pr(Y=v)-\frac{1}{|V|}\right| < \frac{\lambda}{d}\cdot\sqrt{K-1}$$

Proof: Let $N\stackrel{\text{def}}{=}|V|$, and x denote the N-dimensional probability vector defined by X (i.e., $x_i\stackrel{\text{def}}{=}\Pr(X=i)$). Let A denote the Markov process defined by traversing a uniformly selected edge in G; namely, the matrix A is the adjacency matrix of the graph G, normalized by division by d. Denote the eigenvalues of A by $\lambda_1,...,\lambda_N$, and note that $\lambda_1=1$ and $|\lambda_i|\leq\frac{\lambda}{d}$, for every $i>1$. We consider the orthogonal eigenvector basis, $e_1,...,e_N$, where $e_ie_i^\top=\frac{1}{N}$ for each i, and $e_1=(\frac{1}{N},...,\frac{1}{N})$. Again, we write each vector as a linear combination of the vectors in this basis. Denote by c_i the coefficient of x in the direction of e_i (whereas the x_i's are the coefficients in the Kroniker basis). We start by bounding $\sum_i c_i^2$ as follows

$$
\begin{aligned}
\sum_{i=1}^{N} c_i^2\frac{1}{N} &= (\sum_{i=1}^{N} c_ie_i^\top)\cdot(\sum_{i=1}^{N} c_ie_i^\top)^\top\\
&= x\cdot x^\top\\
&= \sum_{i=1}^{N} x_i^2
\end{aligned}
$$

$$
\begin{aligned}
&\leq \max_{\substack{v_1,\ldots,v_N \text{ s.t.} \\ \forall i\ v_i \in [0,(K/N)] \\ \sum_i v_i = 1}} \left\{ \sum_i v_i^2 \right\} \\
&= \frac{N}{K} \cdot \left(\frac{K}{N} \right)^2
\end{aligned}
$$

getting $\sum_{i=1}^N c_i^2 \leq K$. It is also easy to see that $c_1 = 1$ (since e_1 is the only non-zero-sum vector and the sum of its entries equals 1, just as $\sum_i x_i$). Thus, we have

$$
\sum_{i=2}^{N} c_i^2 \leq K - 1 \tag{10.1}
$$

We now consider the differences vector, denoted z, representing the deviation of the random variable Y from the uniform distribution.

$$
\begin{aligned}
z^\top &\stackrel{\text{def}}{=} Ax^\top - e_1^\top \\
&= A(\sum_i c_i e_i)^\top - e_1^\top \\
&= \sum_{i>1} \lambda_i c_i e_i
\end{aligned}
$$

Recall that the lemma claims an upper bound on the norm-1 of z. Instead, we start by providing a bound on its norm-2:

$$
\begin{aligned}
\sum_i z_i^2 &= z \cdot z^\top \\
&= \sum_{i>1} \lambda_i^2 c_i^2 \cdot e_i e_i^\top \\
&\leq \left(\frac{\lambda}{d} \right)^2 \sum_{i>1} c_i^2 \cdot \frac{1}{N} \\
&\leq \left(\frac{\lambda}{d} \right)^2 \frac{K-1}{N}
\end{aligned}
$$

where the last inequality is due to Eq. (10.1). Maximizing the sum of the $|z_i|$'s, subject to the above bound, the lemma follows. (In other words, $\sum_{i=1}^N |z_i| \leq \sqrt{N} \cdot \sqrt{\sum_{i=1}^N z_i^2}$, which in this case is bounded by $\frac{\lambda}{d} \cdot \sqrt{K-1}$). ■

Lecture 11

Square roots in Z_p^* and Primality Testing

Lecture given by Shafi Goldwasser
Notes taken by Yehuda Hassin, Lior Noy and Ben Stein

Summary: In this lecture we introduce two randomized algorithms in number theory. The first algorithm finds in expected polynomial time the square root of a number in Z_p^* for any prime number p. We then use this algorithm to construct a polynomial time algorithm with two-sided errors for primality testing.

11.1 Introduction

Primality testing is known as one of the first areas for the use of randomized algorithms. Moreover, from theoretic point of view this example is still very important. Many of the problems that have randomized algorithms have also a deterministic versions. Hence, the use of randomized algorithms is usually done just for simplicity reasons or as a building stone for deterministic ones. However, there is still no deterministic polynomial time algorithm for primality testing, although we know of some randomized polynomial time algorithms for this problem for twenty five years.

First we introduce some simple definitions and basic facts in group theory.

As a first step towards primality testing we introduce an algorithm $SQRT$ for finding square roots in Z_p^*. The algorithm $SQRT$ is a *Las Vegas* algorithm, i.e., an expected polynomial time algorithm with no error probability. We then convert $SQRT$ into a *Monte Carlo* algorithm, i.e., an algorithm that runs in polynomial time but has an error probability. We use randomization in this algorithm in order to find a non square number in Z_p^*. Although **half** of the numbers in Z_p^* have no square roots, no deterministic algorithm that finds such a number is known.

We then construct an algorithm that tests weather a given number n is a prime number. Let $n = \prod_{i=1}^{k} q^{\alpha_i}$. We rely on the fact that the number of square roots for a square in Z_n^* is 2^k. We then invoke algorithm $SQRT$ to find a square root. Choosing a random solution to this equation and using the above fact (that for composite, not a prime power, there are many solutions), we introduce a two-sided error randomized algorithm for primality testing.

Although the above algorithm does not give us an absolute proof for prime numbers, in the last part of the lecture we will prove that there is a short proof, and hence the language of prime numbers is in $\mathcal{NP}$.

11.2 Definitions and Simple Facts

11.2.1 Definitions

We start with a definition of a group. A group G is a set of elements with a binary operation that satisfies the following properties: The binary operation is associative, has an identity element e, (i.e., for every $a \in G$, $a \circ e = a = e \circ a$), and for every $a \in G$ there exists an inverse element, denoted a^{-1}, such that $a \circ a^{-1} = e$.

Example: The additive group $Z_p = \{0, 1, 2, ..., p-1\}$, when p is an integer, with the binary operation: $a + b \stackrel{\text{def}}{=} (a+b) \bmod p$. By definition, Z_p is a group because

1. $[(a+b) \bmod p + c] \bmod p = [a + (b+c) \bmod p)] \bmod p$ (association).
2. $(a+0) \bmod p = a = (0+a) \bmod p$ (the identity element is 0).
3. $[a + (p-a)] \bmod p = 0 = [(p-a)+a] \bmod p$ (the inverse element of a is $p-a$).

Definition 11.1 Group order: *The order of a group G, denoted by $|G|$, is the number of elements in G.*

Denote by h^m the m-wise multiplication $h \circ \cdots \circ h$.

Definition 11.2 Element order: *The order of an element h in a group G is the smallest integer number m such that $h^m = 1$.*

Definition 11.3 Cyclic group: *A Cyclic group is a group that contains an element g such that $G = \{g, g^2, \ldots, g^{m-1}, g^m\}, m = |G|$, and g^m is the identity of the group. The element g is called the* generator *of the group.*

Example: The group Z_p is a cyclic group because $g = 1$ is a generator of the group:

$$
\begin{aligned}
(1) \bmod p &= 1 \\
(1+1) \bmod p &= 2 \\
(1+1+1) \bmod p &= 3 \\
&. \\
&. \\
&. \\
\underbrace{(1+1+...+1)}_{p-1} \bmod p &= p-1 \\
\underbrace{(1+1+...+1+1)}_{p} \bmod p &= 0
\end{aligned}
$$

Another example of a group that we use frequently is the group Z_p^*.

Let $Z_p^* = \{1 \le x \le p \mid \gcd(x,p) = 1\}$ with the binary operation being multiplication mod p, i.e., $a * b \stackrel{\text{def}}{=} (a * b) \bmod p$.

Definition 11.4 *The order of Z_p^* (the number of elements in Z_p^*) is known as the* Euler Totient Function*: $|Z_p^*| = \phi(p)$.*

Proposition 11.2.1 *If p is a prime number then $|Z_p^*| = p - 1$.*

Proof: Since for a prime p, for every $1 \le x \le p-1$, we know that $\gcd(x,p) = 1$. It follows simply from the definition that $|Z_p^*| = p - 1$. ■

11.2.2 Some facts about Z_p^*

1. From a computational point of view, for every $a \in Z_p^*$ there is a polynomial time algorithm that finds $b \in Z_p^*$ such that $(a * b) \bmod p = 1$. This can be done, for example, by looking at the output of Extended-gcd(a,p): for $a \in Z_p^*$ the output is two integers α and β such that $\alpha p + \beta a = \gcd(a,p) = 1$. Reducing this equation mod p we get that $\beta a \equiv 1 \bmod p$, and so β is the multiplicative inverse of a.

2. *Euler's theorem:* for every $a \in Z_p^* : a^{\phi(p)} \equiv 1 \bmod p$.

3. *Fermat's theorem:* when p is prime, for every $a \in Z_p^* : a^{(p-1)} \equiv 1 \bmod p$.

   Remark: if the converse of the Fermat's sentence was true, then one could suggest that when we want to test if p is prime, we should check weather, for every $a \in Z_p^*$, it holds that $a^{(p-1)} \equiv 1 \bmod p$. But, the converse of Fermat's theorem is not true and there are numbers, known as the Carmichael numbers, such that $a^{(p-1)} \equiv 1 \bmod p$ for every $a \in Z_p^*$, but p is *not* prime. Therefore, we can not use Fermat's theorem as a primality test.

4. When p is a prime, Z_p^* is a cyclic group of order $p-1$, meaning, there exist an element $g \in Z_p^*$ such that $Z_p^* = \{g^1, g^2, \ldots, g^{p-1}\} = \{g^i : i = 1, 2, \ldots, p-1\}$.

Example: $Z_7^* = \{1,2,3,4,5,6\} = \{3,2,6,4,5,1\} = \{3^1 \bmod 7, 3^2 \bmod 7, 3^3 \bmod 7, 3^4 \bmod p, 3^5 \bmod 7, 3^6 \bmod 7\}$, so 3 is a generator of Z_7^*.

Remark: We can see that for every element $a \in Z_p^*$ and a generator g of it, there exists a unique index i such that: $a \equiv g^i \bmod p$. This i is called the *index* of a with respect to g.

11.3 Finding a Square Root

11.3.1 Description of the Problem

Let $a \in Z_p^*$. Given the equation

$$a \equiv x^2 \bmod p$$

is there a solution to this equation?

If a solution x exists such that the above equation is true, a is called *quadratic residue mod p* or a *square mod p*.
If no such solution exists then a is called *quadratic non-residue mod p* or a *non-square mod p*.

What do we know about the number of solutions for such an equation? This depends whether p is a prime or not. If p is prime the following fact is known:

Fact 11.3.1 *Given a prime p, the equation $a \equiv x^2 \bmod p$ has either no solutions, or exactly 2 solutions: x and $-x$.*

Proof: If the equation has no solutions we are done. Assume there is at least one solution x, then clearly $-x$ is also a solution. We'll show that x and $-x$ are the only possible solutions. Let y be another solution, i.e., $a \equiv y^2 \bmod p$ iff $y^2 \equiv a \bmod p$. Since x is a solution, we also have $a \equiv x^2 \bmod p$ iff $x^2 \equiv a \bmod p$, so we can write

$$\begin{aligned} x^2 &\equiv y^2 \bmod p \\ x^2 - y^2 &\equiv 0 \bmod p \\ (x-y)(x+y) &\equiv 0 \bmod p. \end{aligned}$$

Because of the primality of p, either $y = x$ or $y = p - x = -x$. ■

We will later use this fact to discriminate between *primes* and *non-primes*. So, our next task should be to find a way to know if a given a is a *square mod p*. The following fact will be helpful:

Fact 11.3.2 *Given a prime p and a generator g for Z_p^*, then $a \in Z_p^*$ is* square mod p *iff the index of a w.r.t. p and g is* even.

Proof:
($\Leftarrow$ even-index imply square:) Let a be an element with an even index, i.e., for some j it holds that $a \equiv g^{2j} \bmod p \equiv (g^j)^2 \bmod p$. This implies that a is a *square mod p*, with a square root g^j.
($\Rightarrow$ square imply even-index:) a is a square-mod-p, so exist $y \in Z_p^*$ s.t. $a \equiv y^2 \bmod p$. Since g is the generator, g generates every member of Z_p^* and specifically y, so for some t, $y \equiv g^t \bmod p$. Plugging the last equation into $a \equiv y^2 \bmod p$ we get: $a \equiv ((g^t \bmod p)^2) \bmod p \equiv g^{2t} \bmod p$. Since the index of a w.r.t g is unique we get that: $i \equiv 2t \bmod (p-1)$, and the index of a is even (because p is prime and hence odd). ■

Example: we saw before that 3 is a generator of Z_7^*, and we note that elements with even index are indeed square-mod-p. For example, notice first that the element 4 has an even index: the index is 4, since $4 \equiv 3^4 \bmod 7$. 4 is a square, with two roots: the obvious 2 and 5, since $4 \equiv 5^2 \bmod 7$. Note that $5 \equiv -2 \bmod 7$ so we see that Fact 11.3.1 also holds.

Due to the fact that half of the elements are generated with an even index, we can conclude the next corollary:

Corollary 11.5 *Let p be a prime then half of the elements of Z_p^* are squares-mod-p.*

Remark: keeping in mind our ultimate goal for this class, which is to test whether a number is prime or not, and that our current task is to find if an element of Z_p^* is a square-mod-p, one can suggest the following method to find if a is a square: find the index i of a such that $a = g^i$. If i is even, conclude that a is square-mod-p, else conclude that it is not. This simple idea crushes on the shores of reality due to the fact that finding the index of an element, known as the *discrete log problem* is considered a computationally hard task: that is, no efficient algorithm is known for this task. The next claim will present another way to find out if a is square-mod-p.

Definition 11.6 *Given a prime p and $a \in Z_p^*$, the Legendre symbol $\left(\frac{a}{p}\right)$ is defined as*

$$\left(\frac{a}{p}\right) = \begin{cases} 1 & \text{a is a square-mod-p} \\ -1 & \text{a is not a square-mod-p} \end{cases}$$

Theorem 11.7 *Given a prime p and $a \in Z_p^*$,*

$$a^{\frac{p-1}{2}} = \left(\frac{a}{p}\right).$$

This is called the *Euler Criteria*.

To prove the claim, first observe that the following fact is correct:

Fact 11.3.3 *For a prime p, and a* generator *g of Z_p^*, it holds that $g^{\frac{p-1}{2}} = -1$.*

Proof: Look at $\left(g^{\frac{p-1}{2}}\right)^2 = g^{p-1} = 1$ (the last equality holds for every $a \in Z_p^*$). So by Fact 11.3.1 $g^{\frac{p-1}{2}}$ is either 1 or -1. We'll show that 1 can not be a solution. If $g^{\frac{p-1}{2}} = 1$, we have found an index j smaller then $p-1$ s.t. $g^j = g$. If such index exist g can not generate all the elements in Z_p^*, but since we know that g is a generator, the assumption that $g^{\frac{p-1}{2}} = 1$ can not be true, and we get that $g^{\frac{p-1}{2}} = -1$. ■

Example: we saw that 3 is the generator of Z_7^*. Computing $3^{\frac{7-1}{2}} \bmod 7 = 3^3 \bmod 7 = 27 \bmod 7 \equiv -1$ we see that the fact holds.

We can now prove the Euler Criteria:

Proof (of Theorem 11.7): If a is a square-mod-p, then by Fact 11.3.2 a has an even index or in other words $a = g^{2j}$ for some j. Hence, $a^{\frac{p-1}{2}} = g^{2j\frac{p-1}{2}} = g^{j(p-1)} = (g^{p-1})^j = 1^j = 1 = \left(\frac{a}{p}\right)$.

If a is not a square-mod-p, then by Fact 11.3.2 a has an odd index or in other words $a = g^{2j+1}$ for some j. So,

$$\begin{aligned} a^{\frac{p-1}{2}} &= \left(g^{2j+1}\right)^{\frac{p-1}{2}} = g^{[j(p-1)+\frac{p-1}{2}]} = g^{j(p-1)}g^{\frac{p-1}{2}} \\ &= (g^{(p-1)^j})(-1) = 1^j(-1) = -1 = \left(\frac{a}{p}\right) \end{aligned}$$

■

11.3.2 An Algorithm For Finding the Square-Root (mod p)

Now that we have an easy criterion to find out if a is a square-mod-p, we can develop an algorithm for finding the square-root, i.e., an x which satisfy $a \equiv x^2 \bmod p$. This algorithm will give us an efficient way to compute the value of the square-root if it exists.

The input to the Square-Root Algorithm will be prime p and a quadratic-residue $a \in Z_p^*$. The output will be a number x such that $a \equiv x^2 \bmod p$. The algorithm operates differently in two cases, depending on p, one case is easy and the other is more complex.

The two cases For any number p, the value of $p \bmod 4$ is in $\{0,1,2,3\}$. For p that is prime, the possible values are only 1 or 3, since 0 and 2 imply that p is even and hence not a prime. This constitute our two cases:

$$\begin{aligned} &(A) \quad p \bmod 4 = 3 \\ &(B) \quad p \bmod 4 = 1 \end{aligned}$$

(A) $p \bmod 4 \equiv 3$. We can write p as $4t+3$ for some t. We assume that a is a square-mod-p so starting from Theorem 11.7, we develop the following sequence of equivalent conditions:

$$\begin{aligned} \left(\frac{a}{p}\right) &\equiv 1 \bmod p \\ a^{\frac{p-1}{2}} &\equiv 1 \bmod p \\ a^{\frac{4t+3-1}{2}} &\equiv 1 \bmod p \\ a^{2t+1} &\equiv 1 \bmod p \end{aligned}$$

$$\begin{aligned} a^{2t+2} &\equiv a \bmod p \\ a^{2(t+1)} &\equiv a \bmod p \\ \left(a^{t+1}\right)^2 &\equiv a \bmod p. \end{aligned}$$

So we conclude that $x = a^{(t+1)}$ is the square-root of a.

(B) $p \bmod 4 \equiv 1$. Trying to follow the same way as in (A), let us write $p = 4t+1 \Rightarrow a^{\frac{p-1}{2}} = a^{\frac{4t}{2}} = a^{2t} = 1 \bmod p$. But here we hit a wall: multiplying both sides by a as before leaves us with an *odd* power on the left side, and not an even one to get the square-root. Our goal is therefor to have an odd power before the multiplication.

A promising direction to achieve this goal is to write the power of a as $2t = 2^i(2j+1)$. We will first show an "idea" to use this representation to get an odd power in the left side. This "idea" has a problem, but soon we will see how to tackle this problem to get a full solution.

Basic Idea

$$\begin{aligned} a^{\frac{p-1}{2}} &\equiv 1 \bmod p \\ a^{2t} &\equiv 1 \bmod p \\ a^{2^l(2j+1)} &\equiv 1 \bmod p \text{ (take square-root of both sides)} \\ a^{2^{(l-1)}(2j+1)} &\equiv 1 \bmod p \text{ (continue with the square-root until...)} \\ \cdots & \\ a^{2j+1} &\equiv 1 \bmod p \text{ (multiple both side with } a) \\ a^{2j+2} &\equiv a \bmod p \\ \left(a^{j+1}\right)^2 &\equiv a \bmod p \end{aligned}$$

$\Rightarrow x = a^{j+1}$ is the square-root of a.

We have implicitly assumed that taking square roots of $a^2 = 1$ we get $a = 1$. But it may be that $a = -1$. Indeed, what if in one of the square-root operations, the result for the right side is -1 and not 1? In this case we will need an extra "rescue equation". This equation uses a new element b, which is a non-square-mod-p, and hence according to the Euler criteria, $b^{2t} = b^{\frac{p-1}{2}} \equiv -1 \bmod p$. Using this we will manipulate the equation to get an odd power in the left side just before the final step.

The Full Solution We will follow the steps of the basic idea. As long as taking a square-root on both sides results with a 1 in the right side we continue. If the operation results with a -1 (i.e., we reach $a^{2^{(l-1)}(2j+1)} = -1$), we guess a b, and verify that it is indeed non-square with the Euler Criteria. Since $b^{2t} = -1$ we will multiple both side of the 'bad' equation with b^{2t} and get $b^{2t} \cdot a^{2^{(l-1)}(2j+1)} \equiv 1 \bmod p$, and then continue with the sequence of square-root operations. After i times of taking square-root we will get something like:

$$\begin{aligned} b^{\frac{2t}{2^{i_1}}} b^{\frac{2t}{2^{i_2}}} \cdots b^{\frac{2t}{2^{i_k}}} a^{2j+1} &\equiv 1 \bmod p \text{ [for some } i_s\text{'s]} \\ b^{(2^{l_1}+2^{l_2}+\cdots+2^{l_k})(2j+1)} a^{(2j+1)} &\equiv 1 \bmod p \text{ [for } l_s = l - i_s\text{'s, using } 2t = 2^l(2j+1) \text{].} \end{aligned}$$

Observe that the power of b is even. This is true since for each $1 \le s \le k$, l_s is positive (at least in the first time of taking square root we take square root from a^{2t} and not from the b term), the power of b is just a sum of the 2^{l_s}'s (multiplied by $2j+1$) and hence is an even number. So writing the power of b as $2D$, for some D, we finally have:

$$\begin{array}{rcl}
b^{2D}a^{2j+1} & \equiv & 1 \bmod p \text{ (multiple both side with } a\text{)} \\
b^{2D}a^{2(j+1)} & \equiv & a \bmod p \\
\left(b^{D}a^{j+1}\right)^2 & \equiv & a \bmod p \\
\Rightarrow & \text{(the square-root of } a \text{ is:)} & b^{D}a^{j+1}
\end{array}$$

Algorithm SQRT(a,p)

Input: an odd prime p, and a square-mod-p element a
Output: The element x such that $a \equiv x^2 \bmod p$

A) If $p \bmod 4 = 3$, find t such that $p = 4t+3$.
OUTPUT $a^{t+1} \bmod p$.

B) If $p \bmod 4 = 1$, find t such that $p = 4t+1$.

- B.1) Repeatedly choose uniformly $b \in Z_p^*$, until $b^{\frac{p-1}{2}} \bmod p \equiv -1$
- B.2) initialize $i = 2t$ (the power of a), $k = 0$ (the power of b)
- B.3) REPEAT until i is odd
  - B.3.1) $i \leftarrow \frac{i}{2}$, $k \leftarrow \frac{k}{2}$
  - B.3.2) IF $a^i b^k \equiv -1 \bmod p$ THEN $k \leftarrow k + 2t$
- B.4) OUTPUT $a^{\frac{i+1}{2}} b^{\frac{k}{2}}$

Figure 11.1: Algorithm SQRT(a,p)

Comment: There is no known efficient deterministic algorithm to perform step B.1 (above), i.e., to find a non-square element in Z_p^*. However, the Extended Riemann Hypothesis (ERH) implies that there is a non-square element in the first $\log^2 p$ numbers of Z_p^*. Hence, this implies a deterministic algorithm, since we can check all the numbers less than $\log^2 p$ and find a non-square.

Correctness of SQRT The correctness of the algorithm follows from the above discussion. Specifically, Stage $B.3.1$ performs the square root operation by dividing the current power of a (which is i) and b (which is k) by 2. As we explained if the square root is -1 we multiply both sides of the equation by -1; this is done in Stage $B.3.2$ by increasing the power of b by $2t$ (which is the same as multiplying by $b^{2t} \equiv -1$).

Analysis of the running time of SQRT It is easy to see that all the deterministic steps run in time which is polynomial with the description of p. What about the random step $B.1$? Since we saw before that half of the elements of Z_p^* are non-square-mod-p, we have a probability of $\frac{1}{2}$ to choose a 'good' (i.e. non-square) element. The *expected* running time is therefore 2 tries for this step, and polynomial-time for all the algorithm.

Comment: the SQRT algorithm described above never fails, but can take in the worst case infinite time, due to a sequence of bad choices in step B.1. It is therefore a *Las-Vegas* type algorithm. The next section describes how it can be turned into a *Monte-Carlo* type algorithm.

11.3.3 A Monte-Carlo version of the SQRT algorithm

We next provide a monte carlo algorithm $MSQRT(a, p, s)$ for finding square root. The running time is no longer expected polynomial time but rather a polynomial time algorithm. Although, we introduce a failure probability 2^{-s}. The algorithm is different from the last one only in stage B.1. Instead of choosing repeatedly until finding a non-square, we stop after s failures, making sure that the algorithm will always stop, and succeeds with high probability.

The Monte-Carlo version will always run in polynomial-time. For every a and p the algorithm output "FAIL" only when step B.1 did not manage to find a 'good' b in s tries. ¿From Corollary 11.5 the probability for a failure in one try is $\frac{1}{2}$, the probability to FAIL is $\left(\frac{1}{2}\right)^s = 2^{-s}$.

11.4 Randomized Algorithm for Primality Testing

The first randomized algorithms for primality testing were given in 1976-77 by Solovay and Strassen and by Miller and Rabin. Both algorithms have only one-side error. We will describe an algorithm given by Silvio Micali. This algorithm is two-sided error but has a simple analyses.

In order to introduce an algorithm that will separate between prime numbers and composite numbers, we need first to find a mathematical property that separates them. This property is needed to be "easy" for being inspected. The property that our algorithm is going to use is the number of solutions to the equation $x^2 \equiv a \bmod n$. By Fact 11.3.1 we know that if n is a prime number then the number of solutions to the above equation is (0 or) 2. Let $n = \prod_{i=1}^{k} p_i^{\alpha_i}$. We next show that if n is odd and not a prime power (i.e., $k > 1$), then the number of solutions to the above equation is 0 or 2^k.

In order to prove this fact, recall the Chinese remainder theorem (CRT).

Theorem 11.8 *Let $n = n_1 n_2 \cdots n_k$, where the n_i are pairwise relatively prime. For any sequence of residues $r_1 \in Z_{n_1}, \ldots, r_k \in Z_{n_k}$, there is a unique $r \in Z_n$ such that*

$$r \equiv r_i \bmod n_i \quad (for \quad 1 \le i \le k).$$

Moreover r can be computed in polynomial time.

We next prove the theorem regarding the number of solutions to the equation $x^2 \equiv a \bmod n$.

Theorem 11.9 *Let $n = \prod_{i=1}^{k} q_i^{\alpha_i}$ be the factorization of an odd number that is not a prime or a prime power. The number of solutions to the equation $x^2 \equiv a \bmod n$ is either 0 or 2^k.*

Proof: Let $n_i = q_i^{\alpha_i}$ and $a_i \equiv a \bmod n_i$. Then by the CRT we know that being equal to $a \bmod n$ is exactly as being equal simultaneously to $a_i \bmod n_i$ for all i's. But Fact 11.3.1 can be generalized to a prime power such that the equation $x^2 \equiv a_i \bmod n_i$ has either 0 solutions or 2 solutions. If there exists some i with 0 solutions then there is 0 solutions to the equation $x^2 \equiv a \bmod n$ (since if there is any solution to this equation it solves also the equation $x^2 \equiv a_i \bmod n_i$). On the other hand, if every equation $x^2 \equiv a_i \bmod n_i$ has 2 solutions $\{y_i^1, y_i^2\}$ then by using again the CRT there is a unique and different $y \in Z_n$ that corresponds to any one of the 2^k choices of simultaneous solutions for the equations (choosing an array of length k each cell with two choices). ■

We next present the algorithm Primality-Test. The algorithm first chooses uniformly at random solution to the equation $x^2 \equiv a \bmod p$. This is done by choosing x at random and then squaring it. Then we ask algorithm $MSQRT(a, p, s)$ to find a sqrt to a. If p is prime then the only solutions are $\{x, -x\}$ and, with probability $1 - 2^{-s}$, algorithm $MSQRT(a, p, s)$ finds this solution. But if p is not a prime then even if algorithm $MSQRT(a, p, s)$ returns a square root with good probability, it may happen to be *not* in $\{x, -x\}$, but rather be another solution (one of the other $2^k - 2$ possible solutions). The algorithm Primality-Test is presented formally in Figure 11.2.

Algorithm Primality-Test.
Input: a number p and a safety parameter s.
Output: "Prime" or "Composite".

1. If p is either even or a prime power return "Prime".
2. Choose $x \in Z_p^*$ uniformly at random.
3. If $gcd(x, p) \neq 1$ return "Composite".
4. Compute $a = x^2 \bmod p$.
5. $val \leftarrow MSQRT(a, p, s)$.
6. If val equals "FAIL" or $val \notin \{x, -x\}$ or $val^2 \neq a \bmod p$ then return "Composite",
7. Else (i.e., $val \in \{x, -x\}$) return "Prime".

Figure 11.2: Algorithm Primality-Test

Theorem 11.10 *Algorithm* Primality-Test(p, s) *is a monte carlo algorithm that runs in polynomial time and distinguish between a prime number and a composite number. If p is prime then the algorithm outputs "prime" with probability at least* $1 - 2^{-s}$ *and if p is composite then the algorithm outputs "composite" with probability at least* $\frac{1}{2}$.

Proof: We first analyze the running time of the algorithm. The first step is implemented in polynomial time, since one can check if a number p is a power just by trying all the possible powers $k \leq \log p$ and finding a possible solution on the integers by a simple binary search (each time dropping half of the possible integer solutions). Then we use Euclid algorithm and algorithm $MSQRT(a, p, s)$, which are both polynomial time algorithms.

We next prove two claims regarding the two sided errors.

Claim 11.4.1 *If p is a prime number then the probability that algorithm* Primality-Test(p, s) *output "Composite" is less than* 2^{-s}.

Proof: If p is a prime number than algorithm $MSQRT(a,p,s)$ returns the unique solutions $\{x,-x\}$ with probability at least $1-2^{-s}$. ■

Claim 11.4.2 *If p is composite then the probability that algorithm* Primality-Test(p,s) *outputs "Prime" is at most $\frac{1}{2}$.*

Proof: If p is composite then we can guarantee nothing on the output of $MSQRT(a,s)$. But even if this algorithm outputs a square root of a then by Theorem 11.9 it is one of 2^k possible solutions, where k is the number of primes that divides p. Since $k \geq 2$, the probability that the output of $MSQRT$ is in $\{x,-x\}$ is at most $\frac{2}{2^k} \leq \frac{1}{2}$. It follows that if p is composite then the algorithm outputs "prime" with probability at most $\frac{1}{2}$. ■

Combining the two claims, the theorem follows. ■

11.5 Short proof for prime numbers

Let **PRIME** be the set of all prime numbers and let **COMPOSITE** denote the complement set. It is easy to see that **COMPOSITE**$\in \mathcal{NP}$, since the witness for a number n is simply two integers p and q such that their product is n (and none is 1). Computing a product of two numbers is polynomial in the *length* of the numbers. We emphasize that checking if the $\mathcal{NP}$ witness is good must be polynomial in the length of n which is $\log n$. Our goal in this section is to show that there exists short proofs also for **PRIME**.

In first glance this result is somehow surprising since **PRIME** is a set that is a co$\mathcal{NP}$ kind (i.e., for p in the set one must prove that for **every** $x < p$, x does not divide p). However, Pratt discovered a short proof for **PRIME**. Since **COMPOSITE** is also in $\mathcal{NP}$ this means that **PRIME** $\in \mathcal{NP} \cap \mathrm{co}\mathcal{NP}$ hence it is believed that **PRIME** is not $\mathcal{NP}$-COMPLETE.

Again we are looking for a mathematical property that separates the prime numbers from the composite. By Corollary 11.2.1 we know that Z_p^* has $p-1$ elements if p is a prime number, otherwise it has less elements. The first idea to use this characterization (of the primes) is to use the fact that Z_p^* has a generator g. Given such a g we just have to check that the order of g is $p-1$. It seems that we did nothing because this can be checked by looking at the set $\{g^1, g^2, \ldots, g^{p-1}\}$ and verifying that the only power that equals 1 is $p-1$. But the size of this set is exponential in the length of p. In order to solve this problem we use Fact 11.5.1. From this fact we conclude that it suffices to check a set of size at most $O(\log p)$ (the number of distinct primes that divides p) in order to guarantee that g is a generator of Z_p^* and that the order of g is $p-1$.

Fact 11.5.1 *Let p be a prime then g is a generator of Z_p^* iff for every prime q that divides $p-1$ it holds that $g^{\frac{p-1}{q}} \neq 1$.*

Proof: If g is a generator and p is prime clearly its order is $p-1$ hence any power of g less than $p-1$ does not equals 1. Conversely, if g is not a generator then the set $\{g^1, g^2, \ldots, g^{ord(g)}\}$ is a subgroup of Z_p^* and thus divides $p-1$. But if $ord(g) < p-1$ and divides $p-1$ it must also divides $\frac{p-1}{q}$ for some prime q. Hence we conclude that $g^{\frac{p-1}{q}} \equiv 1 \bmod p$. ■

But given the prime factorization of $p-1 = \prod_{i=1}^{k} q_i^{\alpha_i}$; how do we know that every q_i is itself a prime? this is done recursively.

Theorem 11.11 *Let p be a prime number and $p-1=\prod_{i=1}^{k} q_i^{\alpha_i}$. The following recursive witness for p,*

$$W(p)=g\circ(q_1,\alpha_1)\circ(q_2,\alpha_2)\circ\cdots\circ(q_k,\alpha_k)\circ W(q_1)\circ W(q_2)\cdots W(q_k)\ ,$$

is of polynomial size in the length of p and can be verified in polynomial time, hence $PRIME\in\mathcal{NP}$.

Proof: In order to check the witness $W(p)$, for the integer p, we first verify that indeed $p-1=\prod_{i=1}^{k} q_i^{\alpha_i}$. Then we check that $g^{\frac{p-1}{q_i}}\neq 1 \bmod p$, for every $1\leq i\leq k$. Finally, we continue by checking recursively that indeed each q_i is a prime by checking all the $W(q_i)$'s. We have already explained why $W(p)$ is a witness for p.

We next prove that it is of length polynomial in the length of p. Open the recursion in a tree fashion and look at the i'th level of the tree. Notice (can be checked recursively) that the product of all the q's in the i'th level is always less than p. Hence their total length is less than twice the length of p. If we add the generators of the i'th level and the α's we get no more than $4\log p$. The number of levels is less than $\log p$ because each time there is at least a division by 2. Multiplying the number of levels with the size of each level completes the claim about the size of the witness.

The time it takes to verify a witness includes mainly checking that $g^{\frac{p-1}{q_i}}\neq 1$ for every $1\leq i\leq k$. Since taking powers is logarithmic in the exponent, we can conclude that each "piece" of the witness needs $O(poly(\log p))$ time to verify, all together polynomial time in $O(\log p)$. ■

Bibliographic Notes

The square root extraction algorithm presented in this lecture is due to Adleman, Manders and Miller [2].

The primality testing algorithm is due to Silvio Micali. Recall that Micali's algorithm uses any algorithm that extracts square root modulo a prime in order to detect that a given number is not a prime (since the algorithm is likely to fail in such a case). This algorithm is analogous to Rabin's reduction [37] of factoring to extracting sqaure roots modulo a composite. The difference is that Micali's algorithm is a randomized reduction of the problem of testing primality to the problem of extracting square root modulo a prime, and since the latter is known to have a probabilistic polynomial-time algorithm the same follows for the former. In contrast, Rabin's reduction is from a problem believed to be hard (i.e., factorization) to a problem that is not known to have a probabilistic polynomial-time algorithm, such that assuming the intractability of the former it follows that so is the latter.

We comment that the primality tester presented above is less efficient than the standard probabilistic polynomial-time algorithms used for this problem; that is, the primality testers of Solovay and Strassen [41] and of Rabin [38]. We mention that the latter algorithms have one-sided error (i.e., they never declare a prime to be composite), whereas Micali's algorithm may err both ways (but all errors occur with small probability). In contrast, errorless algorithms (which with small probability output no verdict) do exist [1] (building upon [24]) but are far more complex.

The construction of NP-certificates for primality (i.e., the fact that the set of primes is in $\mathcal{NP}$) is due to Pratt [36].

Lecture 12

Hitters and Samplers

Notes taken by Amos Korman, Yoav Rodeh, Eran Tromer

Summary: A *sampler* is an oracle Turing machine that estimates the average of any function $f : \{0,1\}^n \to [0,1]$ with bounded deviation and bounded probability of failure. A *hitter* is an oracle Turing machine that, given a boolean function $f : \{0,1\}^n \to \{0,1\}$, finds x s.t. $f(x) = 1$ with a bounded probability of failure if f has value 1 on at least a constant fraction of $\{0,1\}^n$. These are fundamental and useful problems. We consider the randomness and query complexities of hitters and samplers whose running time is polynomially bounded, and recall lower bounds for each. We show several simple constructions for both, as well as improved constructions based on pairwise-independent sample spaces and on random walks on expander graphs. We then show composite constructions whose complexities match the lower bounds.

12.1 Definitions and Lower Bounds

12.1.1 Definitions

Suppose we are given black-box access to a function $f : \{0,1\}^n \to [0,1]$ or $f : \{0,1\}^n \to \{0,1\}$. A problem of interest is to estimate its average value:

$$\bar{f} \stackrel{\text{def}}{=} \sum_{x \in \{0,1\}^n} f(x)$$

An exact solution would require evaluating f at all points, which takes time exponential in n. However, an approximate solution could be quite useful: for instance, given a $\mathcal{BPP}$ algorithm that has error probability bounded by $\frac{1}{3}$, estimating $\bar{f}$ within error smaller than $\frac{1}{6}$ would suffice to obtain a decision with certainty.

By relaxing the requirement further to allow some probability of failure, we obtain the notion of a sampler. Given oracle access to $f : \{0,1\}^n \to [0,1]$ the sampler outputs a value that is within ε of $\bar{f}$ with probability at least $1 - \delta$. A boolean sampler is similar but the domain of f is limited to $\{0,1\}$. Obviously any sampler is a boolean sampler, but the converse is not true.

Throughout the discussion we assume that the values of f are given with at least $\log_2(1/\varepsilon)$ bits of precision (otherwise samplers don't exist), but no more than $\text{poly}(\log(1/\varepsilon), \log(1/\delta), n)$ bits (otherwise merely reading the oracle tape would exceed the polynomial time bound discussed below).

Definition 12.1 (Sampler) *A probabilistic oracle Turing machine M is a* sampler *if for any $n \in \mathsf{N}$, $\varepsilon \in (0,1]$ and $\delta \in (0,1]$ and for every function $f : \{0,1\}^n \to [0,1]$:*

$$\Pr\left[\left|M^f(\varepsilon,\delta,n) - \bar{f}\right| > \varepsilon\right] < \delta$$

Definition 12.2 (Boolean Sampler) *A probabilistic oracle Turing machine M is a* boolean sampler *if for any $n \in \mathsf{N}$, $\varepsilon \in (0,1]$ and $\delta \in (0,1]$ and for every function $f : \{0,1\}^n \to \{0,1\}$:*

$$\Pr\left[\left|M^f(\varepsilon,\delta,n) - \bar{f}\right| > \varepsilon\right] < \delta$$

Another scenario is this: we are given black-box access to $f : \{0,1\}^n \to \{0,1\}$ and are assured that it has value 1 on at least an ε fraction of its domain. The problem is to find one of these values, i.e., to find a representative of the "good set" $f^{-1}(1)$. As before, in the worst case we could check an exponential number of values without succeeding. By allowing some probability of error we obtain the notion of a hitter.

For $f : \{0,1\}^n \to \{0,1\}$ and $a \in \{0,1\}$, denote:

$$f^{-1}(a) \stackrel{\text{def}}{=} \{x \in \{0,1\}^n \mid f(x) = a\}$$

For $\varepsilon \in [0,1]$, denote:

$$F(n,\varepsilon) \stackrel{\text{def}}{=} \left\{f \;\mid\; f : \{0,1\}^n \to \{0,1\},\ |f^{-1}(1)| \geq \varepsilon \cdot 2^n\right\}$$

That is, $F(n,\varepsilon)$ the set of functions that have value 1 on at least an ε fraction of their domain.

Definition 12.3 (Hitter) *A probabilistic oracle Turing machine M is a* hitter *if for any $n \in \mathsf{N}$, $\varepsilon \in (0,1]$ and $\delta \in (0,1]$ and for any function $f \in F(n,\varepsilon)$:*

$$\Pr\left[M^f(\varepsilon,\delta,n) \in f^{-1}(1)\right] > 1 - \delta$$

For both samplers and hitter, the input δ specifies the allowed error probability. The input ε differs: for samplers it specifies the precision required of the answer, and for hitters it specifies a promise made on the input function.

In Subsection 12.2.1, we show that any sampler yields a boolean hitter with related complexity.

12.1.2 Complexity Measures

As suggested above, the motivation for the relaxed definition is the ability to reduce complexity. For any probabilistic oracle Turing machine M (specifically, for samplers, boolean samplers and hitters), we are interested in the following:

Time complexity $T_M(\varepsilon,\delta,n)$: We require $T_M(\varepsilon,\delta,n) = \text{poly}\left(n, \frac{1}{\varepsilon}, \log\left(\frac{1}{\delta}\right)\right)$. That is, M should halt within time polynomial in n, $\frac{1}{\varepsilon}$ and $\log\left(\frac{1}{\delta}\right)$.

Query complexity $Q_M(\varepsilon,\delta,n)$: The maximum number of queries made by $M^f(\varepsilon,\delta,n)$, taken over all possible coin tosses and all possible oracles $f \in F(n,\varepsilon)$. We wish to minimize this, because evaluation may be expensive or the context may bound the number of permissible evaluations.

Randomness complexity $R_M(\varepsilon, \delta, n)$**:** The maximum number of probabilistic bits used by $M^f(\varepsilon, \delta, n)$, i.e., the number of binary coin flips it may use, taken over all possible oracles $f \in F(n, \varepsilon)$. We wish to minimize this parameter as well: for small δ we may need to try many different random choices, so it is useful to reduce the space of this choices (we'll see an example of this in Section 12.4).

12.1.3 Lower Bounds

The following lower bounds on complexity hold:

	$Q_M(\varepsilon,\delta,n) \geq$	$R_M(\varepsilon,\delta,n) \geq$
Hitters	$\Omega\left(\frac{1}{\varepsilon}\log\left(\frac{1}{\delta}\right)\right)$	$n + \log_2\left(\frac{1}{\delta}\right) - \log_2(Q_M(\varepsilon,\delta,n)) - O(1)$
Samplers	$\Omega\left(\frac{1}{\varepsilon^2}\log\left(\frac{1}{\delta}\right)\right)$	$n + \log_2\left(\frac{1}{\delta}\right) - \log_2(Q_M(\varepsilon,\delta,n)) - O(1)$

We proceed to prove the lower bounds for hitters[1]. For proofs of the lower bounds on the complexity of samplers, see [14] or the weaker results of Subsection 12.2.1.

The lower bounds which we derive on the query and randomness complexities of hitters apply only when ε, as a function of n and δ, is upper bounded by some constant $c < 1$. Some limitation must indeed be imposed on ε, since when ε approaches 1 very quickly it's possible that few queries or little randomness are required. For instance, consider the case $\varepsilon = 1 - \frac{1}{2^n}$, i.e., $|f^{-1}(0)| = 1$. In this case, for any fixed δ and sufficiently large n, a *single* random query will hit $f^{-1}(1)$ with probability greater than δ. Similarly, when $|f^{-1}(0)| = 1$ evaluating any two *fixed* elements guarantees success regardless of δ, using no randomness.

For the bounds to hold we also need to restrict the values of δ, as the following shows: for given n and ε, the minimum query complexities is no more than $(1-\varepsilon)\cdot 2^n$, regardless of δ. Similarly, an algorithm that is allowed to make 2^n queries needs no randomness, regardless of δ.

We will need the following technical lemma:

Lemma 12.1.1 *For any constant $c \in (0,1)$, there exists a constant $c' > 0$ such that for any $N, s \in \mathsf{N}$ and $\varepsilon \in (0,c)$ such that $(1-\varepsilon)\cdot N > s$:*

$$\frac{\binom{N-s}{\varepsilon\cdot N}}{\binom{N}{\varepsilon\cdot N}} > e^{-c'\cdot s\cdot\varepsilon}$$

Moreoever, when $\varepsilon \ll 1$, $s \ll N$, $\varepsilon\cdot N \gg 1$ and $(1-\varepsilon)\cdot N - s \gg 1$: $c' \approx 2$.

Proof: According to Stirling's formula: $k! \approx \sqrt{2\pi k}\left(\frac{k}{e}\right)^k$ for $k \gg 1$. This yields the following approximation for the binomial coefficient $\binom{a}{b}$ when b and $(a-b)$ are large:

$$\binom{a}{b} = \frac{a!}{b!(a-b)!} \approx \frac{\sqrt{2\pi a}\left(\frac{a}{e}\right)^a}{\sqrt{2\pi b}\left(\frac{b}{e}\right)^b\cdot\sqrt{2\pi(a-b)}\left(\frac{a-b}{e}\right)^{a-b}} = \sqrt{\frac{a}{2\pi\cdot b\cdot(a-b)}}\cdot\left(\frac{a}{b}\right)^b\cdot\left(\frac{a}{a-b}\right)^{a-b}$$

Denoting $\sigma = \frac{s}{N}$ and cancelling out N factors, we get:

$$\frac{\binom{N-s}{\varepsilon\cdot N}}{\binom{N}{\varepsilon\cdot N}} \approx \frac{\sqrt{\frac{1-\sigma}{N\cdot 2\pi\cdot\varepsilon\cdot(1-\sigma-\varepsilon)}}\cdot\left(\frac{1-\sigma}{\varepsilon}\right)^{\varepsilon\cdot N}\left(\frac{1-\sigma}{1-\sigma-\varepsilon}\right)^{N-s-\varepsilon\cdot N}}{\sqrt{\frac{1}{N\cdot 2\pi\cdot\varepsilon\cdot(1-\varepsilon)}}\cdot\left(\frac{1}{\varepsilon}\right)^{\varepsilon\cdot N}\left(\frac{1}{1-\varepsilon}\right)^{N-\varepsilon\cdot N}} = \frac{\sqrt{\frac{1-\sigma}{(1-\sigma-\varepsilon)}}}{\sqrt{\frac{1}{(1-\varepsilon)}}}\cdot\frac{\left(\frac{1-\sigma}{\varepsilon}\right)^{\varepsilon\cdot N}}{\left(\frac{1}{\varepsilon}\right)^{\varepsilon\cdot N}}\cdot\frac{\left(\frac{1-\sigma}{1-\sigma-\varepsilon}\right)^{N-s-\varepsilon\cdot N}}{\left(\frac{1}{1-\varepsilon}\right)^{N-\varepsilon\cdot N}}$$

[1]The proofs in this section are provided by the scribes and did not appear in the lecture.

Note that $\frac{1-\sigma}{1-\sigma-\varepsilon} > \frac{1}{1-\varepsilon}$, hence for some constants $c_1, c_2 < 1$ and c', we get the lower bound:

$$\frac{\left(\frac{1-\sigma}{\varepsilon}\right)^{\varepsilon\cdot N}}{\left(\frac{1}{\varepsilon}\right)^{\varepsilon\cdot N}} \cdot \frac{\left(\frac{1}{1-\varepsilon}\right)^{N-s-\varepsilon\cdot N}}{\left(\frac{1}{1-\varepsilon}\right)^{N-\varepsilon\cdot N}} = (1-\sigma)^{\varepsilon\cdot N}\cdot(1-\varepsilon)^{s} = (1-\sigma)^{\frac{1}{\sigma}\cdot s\cdot\varepsilon}\cdot(1-\varepsilon)^{\frac{1}{\varepsilon}\cdot s\cdot\varepsilon} > (c_1)^{s\cdot\varepsilon}\cdot(c_2)^{s\cdot\varepsilon} = e^{-c'\cdot s\cdot\varepsilon}$$

where the last inequality relies on ε and $\sigma = \frac{s}{N}$ being upper bounded by constants smaller than 1. The approximations (Stirling's formula) can be accounted for by a small increase of c' (rigorous analysis is possible using strict bounds instead of approximations). The main part of the lemma follows.

Moreover, the additional conditions listed in the lemma imply that the approximations are very accurate, and furthermore $\varepsilon, \sigma \ll 1$ implies $c_1, c_2 \approx \frac{1}{e}$, hence $c' \approx 2$.

■

Theorem 12.4 *For any constant $c \in (0,1)$: for any polynomial time hitter H and any $n \in \mathsf{N}$, $\varepsilon \in (0,c)$ and $\delta \in (0,\ e^{-\varepsilon\cdot(1-\varepsilon)\cdot 2^n})$,*

$$Q_H(\varepsilon,\delta,n) \geq \Omega\left(\frac{1}{\varepsilon}\log\left(\frac{1}{\delta}\right)\right)$$

Proof: Using Yao's Lemma [44], to prove the lower bound on probabilistic algorithms it suffices to show a probability distribution Φ on $F(n,\varepsilon)$ such that every deterministic oracle Turing machine D needs $\Omega\left(\frac{1}{\varepsilon}\log\left(\frac{1}{\delta}\right)\right)$ queries to hit a good element with probability at least δ, where the probability is taken over the distribution Φ.

We define Φ to be the uniform distribution over all function $f \in F(n)$ s.t. $|f^{-1}(1)| = \lceil\varepsilon \cdot 2^n\rceil$.

Let D be a a deterministic oracle Turing machine D that succeeds with probability at least δ. Define $Samples_D(\varepsilon,\delta,n)$ to be the set of all queries made by $D^f(\varepsilon,\delta,n)$ when D receives only "0" answers to its oracle queries. This set depends only on the inputs n, ε and δ, and not on the function f. Let $s = |Samples_D(\varepsilon,\delta,n)|$.

If $s \geq \frac{1}{\varepsilon}\log\left(\frac{1}{\delta}\right)$ then we're done. Otherwise: the limited domain of δ implies $\frac{1}{\varepsilon}\log\left(\frac{1}{\delta}\right) < (1-\varepsilon)\cdot 2^n$, so we have $s < (1-\varepsilon)\cdot 2^n$. The probability that D does not succeed is

$$\begin{aligned}
&\Pr\left[f^{-1}(1) \cap Samples_D(\varepsilon,\delta,n) = \emptyset\right] \\
= &\ \frac{\#\text{ ways to choose } \varepsilon\cdot 2^n \text{ elements outside } Samples_D(\varepsilon,\delta,n)}{\#\text{ ways to choose } \varepsilon\cdot 2^n \text{ elements}} \\
= &\ \frac{\binom{2^n-s}{\varepsilon\cdot 2^n}}{\binom{2^n}{\varepsilon\cdot 2^n}} > e^{-c'\cdot\varepsilon\cdot s}
\end{aligned}$$

for some constant $c' < 1$, where the inequality follows from Lemma 12.1.1 using $N = 2^n$.

The probability $e^{-c'\cdot\varepsilon\cdot s}$ must be smaller than δ, and $s \leq Q_D(\varepsilon,d,n)$ by the definition of query complexity, so we obtain:

$$Q_D(\varepsilon,d,n) \geq s \geq \Omega\left(\frac{1}{\varepsilon}\log\left(\frac{1}{\delta}\right)\right)$$

Note that when the conditions of Lemma 12.1.1 hold (and in particular ε is small), the constant hidden in the Ω notation is approximately 2. ■

Theorem 12.5 *For any polynomial time hitter H and constant $\varepsilon \in (0,1)$ s.t. $Q_H(\varepsilon,\delta,n) < (1-\varepsilon)\cdot 2^n$:*

$$R_H(\varepsilon,\delta,n) \geq n + \log_2\left(\tfrac{1}{\delta}\right) - \log_2(Q_H(\varepsilon,\delta,n)) - O(1)$$

Proof: Fix ε, δ and n. Denote by r the maximum number of random bits used by $H^f(\varepsilon,\delta,n)$, taken over all possible $f \in F(n,\varepsilon)$.

First we show that $\delta \geq \frac{1}{2^r}$. If this does not hold then H answers correctly for any random seed, so by fixing its random choices arbitrarily we obtain a deterministic hitter H' with query complexity $Q_{H'}(\varepsilon,\delta,n) < (1-\varepsilon)\cdot 2^n$. This is impossible since H' fails when given the following oracle $f_{H'}(x) \in F(n,\varepsilon)$:

$$f_{H'}(x) = \begin{cases} 0 & \text{if } H' \text{ queries } x \\ 1 & \text{otherwise} \end{cases}$$

We now turn to the heart of the analysis (of H). For $w \in \{0,1\}^r$, let $\mathit{Samples}(w) \subseteq \{0,1\}^n$ denote the set of all queries made by H with w as its random coin tosses when H receives only "0" anwers to its oracle queries. For $U \subseteq \{0,1\}^r$, define $\mathit{Samples}(U) = \bigcup_{w\in U} \mathit{Samples}(w)$, and define the function $f_U : \{0,1\}^n \to \{0,1\}$ such that

$$f_U(x) = \begin{cases} 0 & \text{if } x \in \mathit{Samples}(U) \\ 1 & \text{otherwise} \end{cases}$$

If the random coin toss of H is $w \in U$ then given f_U as an oracle, H receives only "0" answers and thus fails. Since the coin toss of H is in U with probability at least $\frac{1}{2^r}|U|$, this yields a lower bound on the probability of failure [2]:

$$\Pr\left[H^{f_U} \notin f_U^{-1}(1)\right] > \tfrac{1}{2^r}|U| \tag{12.1}$$

Denote by q the maximum number of samples made by H^f taken over all $f \in F(n,\varepsilon)$ and random coin tosses. Then, for every w, $|\mathit{Samples}(w)| \leq q$. Therefore, $\mathit{Samples}(U) \leq q\cdot|U|$, and so:

$$|f_U^{-1}(1)| = 2^n - |\mathit{Samples}(U)| \geq 2^n - q\cdot|U|$$

Setting $|U| = \left\lfloor \frac{1}{q}\cdot(1-\varepsilon)\cdot 2^n \right\rfloor$, we get

$$|f_U^{-1}(1)| \geq 2^n - q\cdot|U| = 2^n - q\cdot\lfloor\tfrac{1}{q}\cdot(1-\varepsilon)\cdot 2^n\rfloor \geq 2^n - q\cdot\tfrac{1}{q}\cdot(1-\varepsilon)\cdot 2^n = \varepsilon\cdot 2^n$$

Thus, $f_U \in F(n,\varepsilon)$. H is a hitter, so according to (12.1) we have $\frac{1}{2^r}|U| < \delta$, i.e.,

$$\tfrac{1}{2^r}\left\lfloor \tfrac{1}{q}\cdot(1-\varepsilon)\cdot 2^n \right\rfloor \leq \delta$$

Hence:

$$\tfrac{1}{2^r}\left(\tfrac{1}{q}\cdot(1-\varepsilon)\cdot 2^n - 1\right) \leq \delta$$
$$\tfrac{1}{q}\cdot(1-\varepsilon)\cdot 2^n \leq 2^r\cdot\delta + 1 \leq 2^{r+1}\cdot\delta$$

where the last inequality follows from $\delta \geq \frac{1}{2^r}$. After taking logarithms and rearranging:

$$r + 1 \geq n + \log_2(1-\varepsilon) + \log_2\left(\tfrac{1}{\delta}\right) - \log_2(q)$$

ε is constant, so the claim follows. ■

[2] We assume w.l.o.g. that H always output a value which it has queried.

Corollary 12.6 *A hitter H with optimal query complexity has* $R_H(\varepsilon,\delta,n) \geq n+\Omega(\log\left(\frac{1}{\delta}\right))-\log_2\left(\frac{1}{\varepsilon}\right)$

Proof: If we assume the H is optimal with respect to the number of queries it makes, then $Q_H(\varepsilon,\delta,n) = O(\frac{1}{\varepsilon}\log\left(\frac{1}{\delta}\right))$. This gives us:

$$R(H) \geq n + \log_2\left(\frac{1}{\delta}\right) - \log_2(O(\frac{1}{\varepsilon}\log\left(\frac{1}{\delta}\right)))$$

And therefore

$$\begin{aligned} R(H) &\geq n + \log_2\left(\frac{1}{\delta}\right) - \log_2\left(\frac{1}{\varepsilon}\right) - \log_2\log\left(\frac{1}{\delta}\right) - O(1) \\ &= n + \Omega(\log\left(\frac{1}{\delta}\right)) - \log_2\left(\frac{1}{\varepsilon}\right) \end{aligned}$$

■

12.2 Simple Algorithms

In the rest of this lecture, we present several constructions that fulfill the time bound and analyze their query and randomness complexities. This chapter shows three simple constructions.

12.2.1 Constructing a Hitter from a Boolean Sampler

First, we show that any boolean sampler yields a hitter with related complexity. Using a boolean sampler S, we construct the hitter MONITOR-HITTER$_S$ thus:

Algorithm: MONITOR-HITTER$^f_S(\varepsilon,\delta,n)$
Invoke S^f with parameters $(\frac{\varepsilon}{2},\frac{\delta}{2},n)$. Monitor the operation of S^f: if some query x is answered by $f(x)=1$ then output x and halt. If S halts before the above happens, output $\perp$.

Proposition 12.2.1 MONITOR-HITTER$_S$ *is a hitter.*

Proof: Let S be a boolean sampler, let $f \in F(n,\varepsilon)$. Let MONITOR-HITTER$^f_S(\varepsilon,\delta,n)$ be a random variable taken from output probability distribution of MONITOR-HITTER^f_S on input (ε,δ,n). We need to prove that

$$\Pr\left[\text{MONITOR-HITTER}^f_S(\varepsilon,\delta,n) \notin f^{-1}(1)\right] < \delta$$

This is exactly the probability that the invoked sampler $S^f\left(\frac{\varepsilon}{2},\frac{\delta}{2},n\right)$ does not receive any "1" answers on its queries. Define the random variable $s = S^f\left(\frac{\varepsilon}{2},\frac{\delta}{2},n\right)$ to represent the output of S on input $\left(\frac{\varepsilon}{2},\frac{\delta}{2},n\right)$ and access to the oracle f. Then:

$$\begin{aligned} &\Pr\left[S^f \text{ receives only 0's}\right] = \\ &\quad \Pr\left[S^f \text{ receives only 0's and } s < \tfrac{\varepsilon}{2}\right] + \Pr\left[S^f \text{ receives only 0's and } s \geq \tfrac{\varepsilon}{2}\right] \end{aligned} \tag{12.2}$$

Note that $\bar{f} > \varepsilon$ (because $f \in F(n,\varepsilon)$), and since S is a sampler we have:

$$\Pr\left[s < \frac{\varepsilon}{2}\right] < \frac{\delta}{2} \tag{12.3}$$

On the other hand: in all cases where the coin tosses of S caused S to receive only 0's, they will cause exactly the same scenario if f were replaced by the constant zero function f_0 ($\forall x : f_0(x) = 0$), for which $\bar{f}_0 = 0$. Thus:

$$\Pr\left[s \geq \frac{\varepsilon}{2} \text{ and S receives only 0's}\right] \leq \Pr\left[S^{f_0}\left(\frac{\varepsilon}{2}, \frac{\delta}{2}, n\right) \geq \frac{\varepsilon}{2}\right] < \frac{\delta}{2} \tag{12.4}$$

Combining equations (12.2), (12.3) and (12.4) we get:

$$\Pr\left[\text{Monitor-Hitter}_S^f(\varepsilon, \delta, n) \notin f^{-1}(1)\right] = \Pr\left[S^f \text{ receives only 0's}\right] < \delta$$

■

Note that using the lower bound $Q_H \geq \frac{1}{\varepsilon}\log\left(\frac{1}{\delta}\right)$ on the query complexity of hitters, this reduction immediately yields a (non-tight) lower bound of $Q_S \geq \frac{1}{2\varepsilon}\log\left(\frac{1}{2\delta}\right)$ on the query complexity of samplers. Similarly, the $R_H \geq n + \log_2\left(\frac{1}{\delta}\right) - \log_2(Q_H(\varepsilon, \delta, n)) - O(1)$ lower bound on the randomness complexity of hitters implies a lower bound $R_S \geq n + \log_2\left(\frac{1}{2\delta}\right) - \log_2(Q_S(2\varepsilon, 2\delta, n)) - O(1)$ on the randomness complexity of samplers.

12.2.2 A Naive Hitter

The simplest method to obtain a hitter is to sample random points uniformly, and independently, hoping to hit a good one:

Algorithm: Naive-Hitter$^f(\varepsilon, \delta, n)$

Set $q = \frac{1}{\varepsilon}\log\left(\frac{1}{\delta}\right)$.
Randomly choose $x_1, \ldots, x_q \in \{0,1\}^n$, uniformly and independently.
Query f on $x_1, \ldots, x_q$.
If $f(x_i) = 1$ for some i then output the first such x_i. Otherwise output 0.

Proposition 12.2.2 Naive-Hitter *is a hitter.*

Proof: For any $f \in F(n, \varepsilon)$,

$$\Pr\left[\text{Naive-Hitter}^f(n, \varepsilon, \delta) \in f^{-1}(1)\right] = 1 - \Pr\left[\forall i, f(x_i) = 0\right] \geq 1 - (1 - \varepsilon)^q = 1 - \delta$$

■

The complexities of Naive-Hitter are:

- $Q_{\text{Naive-Hitter}}(\delta, \varepsilon, n) = q = \frac{1}{\varepsilon}\log\left(\frac{1}{\delta}\right)$ — this matches the lower bound
- $R_{\text{Naive-Hitter}}(\delta, \varepsilon, n) = q \cdot n = \frac{n}{\varepsilon}\log\left(\frac{1}{\delta}\right)$

We see that by simply choosing an appropriate number of samples at random, we attain optimal (up to a constant multiplicative factor) query complexity. However, the randomness complexity is much higher than the lower bound.

12.2.3 A Naive Sampler

A similar approach yields a sampler (and hence a boolean sampler):

Algorithm: NAIVE-SAMPLER$^f(\varepsilon, \delta, n)$

Set $q = \frac{1}{\varepsilon^2} \log\left(\frac{2}{\delta}\right)$.
Randomly choose $x_1, \ldots, x_q \in \{0,1\}^n$, uniformly and independently.
Query f on $x_1, \ldots, x_q$.
Output $\frac{1}{q} \sum_i f(x_i)$.

Proposition 12.2.3 NAIVE-SAMPLER *is a sampler.*

Proof: Observe that for each x_i, the expected value of $f(x_i)$ is $\bar{f}$. Using Chernoff's bound:

$$\Pr\left[\left|\text{NAIVE-SAMPLER}^f(n, \varepsilon, \delta) - \bar{f}\right| > \varepsilon\right] =$$
$$\Pr\left[\left|\frac{1}{q}\sum_i f(x_i) - \frac{1}{q}\sum_i \mathsf{E}(f(x_i))\right| > \varepsilon\right] < 2e^{-\varepsilon^2 q} = \delta$$

■

The complexities of NAIVE-SAMPLER are:

- $Q_{\text{NAIVE-SAMPLER}}(\varepsilon, \delta, n) = q = \frac{1}{\varepsilon^2} \log\left(\frac{2}{\delta}\right)$ — this matches the lower bound
- $R_{\text{NAIVE-SAMPLER}}(\varepsilon, \delta, n) = n \cdot q = \frac{n}{\varepsilon^2} \log\left(\frac{2}{\delta}\right)$

Similarly to NAIVE-HITTER, the query complexity is optimal (up to a constant multiplicative factor) but the randomness complexity is bad.

12.3 Better Constructions

We have seen that the naive approach requires too much randomness. In previous lectures we have seen approaches for generating objects that are "close to random" in various senses while using few random bits. Using these approaches we can build samplers and hitters with reduced complexity.

12.3.1 A Pairwise-Independent Hitter

First we consider *small pairwise independent samples spaces* (Lectures 3 and 4). For concreteness, consider the construction based on finite fields. We wish to produce pairwise-independent elements from $\{0,1\}^n$, so we can use the field $GF(2^n) \equiv \mathsf{Z}_2[x]/f(x)$ where $f(x)$ is an irreducible polynomial in $\mathsf{Z}_2[x]$. Assuming we need $q \leq 2^n$ elements, the construction yields a sample space of size merely $2n$. That is, only $2n$ bits of randomness are required to get q pairwise-independent n-bit elements.

Algorithm: PAIRWISE-HITTER$^f(\varepsilon, \delta, n)$

Set $q = \frac{1}{\varepsilon \cdot \delta}$.
If $q > 2^n$ then evaluate $f(x)$ for all $x \in \{0,1\}^n$ and output the first $x \in f^{-1}(1)$. Otherwise:
Generate a sequence of q pairwise independent elements $x_1, \ldots, x_q \in \{0,1\}^n$.
Query f on $x_1, \ldots, x_q$.
If for some i, $f(x_i) = 1$, output x_i. Otherwise output $\perp$.

Proposition 12.3.1 PAIRWISE-HITTER *is a hitter.*

Proof: For $q > 2^n$, correctness is obvious. Otherwise, assume $f \in F(n, \varepsilon)$ and define q random variables $\zeta_1, \ldots, \zeta_q$, where

$$\zeta_i = \begin{cases} 1 & \text{if } f(x_i) = 1 \\ 0 & \text{otherwise} \end{cases}$$

We have that $\mathsf{E}(\zeta_i) = \bar{f}$ and $\mathrm{Var}(\zeta_i) = \mathsf{E}({\zeta_i}^2) - \mathsf{E}(\zeta_i)^2 = \mathsf{E}(\zeta_i) - \mathsf{E}(\zeta_i)^2 = \bar{f} \cdot (1 - \bar{f})$. Using Chebychev's inequality:

$$\begin{aligned}
\Pr\left[\text{PAIRWISE-HITTER} \not\in f^{-1}(1)\right] &= \Pr\left[\tfrac{1}{q}\sum_i \zeta_i = 0\right] \\
&\le \Pr\left[\left|\tfrac{1}{q}\sum_i \zeta_i - \bar{f}\right| > \bar{f}\right] \\
&< \frac{\mathrm{Var}(\zeta_i)}{q \cdot \bar{f}^2} = \frac{\bar{f} \cdot (1 - \bar{f})}{q \cdot \bar{f}^2} \\
&< \frac{\bar{f}}{q \cdot \bar{f}^2} = \tfrac{1}{q \cdot \bar{f}} \le \tfrac{1}{q \cdot \varepsilon} = \delta
\end{aligned}$$

■

The complexities of PAIRWISE-HITTER are:

- $Q_{\text{PAIRWISE-HITTER}}(\varepsilon, \delta, n) \le q = \frac{1}{\varepsilon \cdot \delta}$
- $R_{\text{PAIRWISE-HITTER}}(\varepsilon, \delta, n) = 2n$

Note that this randomness complexity is optimal (up to a constant factor) for any value of ε. The query complexity is optimal when δ is constant, but otherwise it is not optimal.

12.3.2 A Pairwise-Independent Sampler

Similarly, we obtain a sampler based on pairwise-independent elements of $\{0,1\}^n$.

Algorithm: PAIRWISE-SAMPLER$^f(\varepsilon, \delta, n)$

Set $q = \frac{1}{\varepsilon^2 \cdot \delta}$.
If $q > 2^n$ then evaluate $f(x)$ for all $x \in \{0,1\}^n$ and output the average. Otherwise:
Generate a sequence of q pairwise independent elements $x_1, \ldots, x_q \in \{0,1\}^n$ using the finite-field construction.
Query f on $x_1, \ldots, x_q$.
Output $\frac{1}{q}\sum_i f(x_i)$.

Proposition 12.3.2 PAIRWISE-SAMPLER *is a sampler.*

Proof: For $q > 2^n$ correctness is obvious. Otherwise, define q random variables $\zeta_1, \ldots, \zeta_q$, where

$$\zeta_i = \begin{cases} 1 & \text{if } f(x_i) = 1 \\ 0 & \text{otherwise} \end{cases}$$

As before, $\mathsf{E}(\zeta_i) = \bar{f}$ and $\mathrm{Var}(\zeta_i) = \bar{f}(1-\bar{f})$. Using Chebychev's inequality:

$$\begin{aligned}\Pr\left[\left|S^f(n,\varepsilon,\delta) - \bar{f}\right| > \varepsilon\right] &= \Pr\left[\left|\frac{1}{q}\sum_i \zeta_i - \bar{f}\right| > \varepsilon\right] \\ &< \frac{\mathrm{Var}(\zeta_i)}{q\cdot\varepsilon^2} = \frac{\bar{f}\cdot(1-\bar{f})}{q\cdot\varepsilon^2} \\ &< \frac{1}{q\cdot\varepsilon^2} = \delta\end{aligned}$$

■

The complexities of PAIRWISE-SAMPLER are:

- $Q_{\text{PAIRWISE-SAMPLER}}(\varepsilon,\delta,n) = q = \frac{1}{\varepsilon^2\cdot\delta}$
- $R_{\text{PAIRWISE-SAMPLER}} = 2n$.

As before, this randomness complexity is optimal (up to a constant factor), whereas the query complexity is optimal only when δ is constant.

12.3.3 An Expander-Walk Hitter

Another approach to producing "random-looking" elements using little randomness is *random walks on expander graphs* (Lectures 9 and 10). For any given ε, we will need a strongly-constructible family of expander graphs $\{G_n\}$ with algebraic expansion coefficient $\lambda \le \frac{\varepsilon}{10}$ and degree $d = \mathrm{poly}\left(\frac{1}{\lambda}\right)$.

This can be obtained directly using the second construction mentioned in Lecture 10 . Alternatively, we can start with any construction that has $d = \mathrm{poly}\left(\frac{1}{\lambda}\right)$ and use amplification to achieve sufficiently low λ (i.e., connect each vertex in G_n to all nodes in its k-neighbourhood, where a suitable value of k is chosen using Theorem 3 in Lecture 10 and Proposition 3.1 in Lecture 9). The relation $d = \mathrm{poly}\left(\frac{1}{\lambda}\right)$ is maintained by the amplification.

Algorithm: WALK-HITTER$^f(\varepsilon,\delta,n)$

Choose a strongly-constructible expander graph G_{2^n} as specified above ($\lambda \le \frac{\varepsilon}{10}$, $d = \mathrm{poly}\left(\frac{1}{\lambda}\right)$).
Set $q = O\left(\frac{1}{\varepsilon}\log\left(\frac{1}{\delta}\right)\right)$ (the constant is determined by the proof, see below).
Generate a sequence of q elements $x_1,\ldots x_q \in \{0,1\}^n$ by conducting a random walk of length q on the expander from a random initial point.
Query f on $x_1,\ldots,x_q$. If for some i, $f(x_i) = 1$, output the first such x_i. Otherwise output 0.

Proposition 12.3.3 WALK-HITTER *is a hitter.*

Proof: Assume $f \in F(n,\varepsilon)$. Denote $B = f^{-1}(0)$.

$$\begin{aligned}&\Pr\left[\text{WALK-HITTER} \notin f^{-1}(1)\right] \\ &\quad = \Pr\left[\text{a walk of length } q \text{ stays in } B\right] \qquad \text{(using Theorem 6 in Lecture 10)} \\ &\quad < \sqrt{\rho(B)}\cdot\left(\sqrt{\rho(B)}+\lambda\right)^{q-1} \\ &\quad \le \sqrt{1-\varepsilon}\cdot\left(\sqrt{1-\varepsilon}+\lambda\right)^{q-1} \\ &\quad < \left(1-\tfrac{1}{3}\varepsilon\right)\cdot\left(1-\tfrac{1}{3}\varepsilon+\tfrac{1}{10}\varepsilon\right)^{q-1} \\ &\quad \le \left(1-\tfrac{\varepsilon}{5}\right)^q = \delta\end{aligned}$$

■

The complexities of WALK-HITTER are:

- $Q_{\text{WALK-HITTER}}(\varepsilon, \delta, n) = q = O\left(\frac{1}{\varepsilon}\log\left(\frac{1}{\delta}\right)\right)$
- $R_{\text{WALK-HITTER}}(\varepsilon, \delta, n) = n + (q-1)\log d = n + O\left(\frac{1}{\varepsilon}\log\left(\frac{1}{\delta}\right)\log\left(\frac{1}{\varepsilon}\right)\right)$

Note that the query complexity is optimal (up to constant factor), but the randomness complexity is optimal only when ε is constant. This is the reverse of the case for the pairwise-independent constructions above.

12.4 Composed Constructions

We have seen:

	Q_M	R_M
PAIRWISE-HITTER PAIRWISE-SAMPLER	optimal for constant δ	optimal
WALK-HITTER	optimal	optimal for constant ε

In this section we combine the two constructions, using the "well-handled" parameter (ε or δ) of one to fix the "badly-handled" parameter of the other. We thus obtain composite constructions that are optimal in all parameters.

12.4.1 A Composed Hitter

PAIRWISE-HITTER uses $2n$ bits of randomness. Consider a deterministic version D-PAIRWISE-HITTER of it, which accepts these $2n$ bits as an additional input. That is,

$$\text{PAIRWISE-HITTER}^f(\varepsilon, \delta, n) \equiv \left[r \in_R \{0,1\}^{2n} \text{ ; return D-PAIRWISE-HITTER}^f(\varepsilon, \delta, n, r)\right]$$

We fix the input δ of D-PAIRWISE-HITTER as $\frac{1}{2}$, since it is not optimal in this parameter. This gives a natural definition of good values of r:

Definition 12.7 (Good Seed) *$r \in \{0,1\}^{2n}$ is called a* good seed *(w.r.t. f, ε, n) if* D-PAIRWISE-HITTER$^f(\varepsilon, \frac{1}{2}, n, r) \in f^{-1}(1)$. *Denote $G \stackrel{\text{def}}{=} \{r \mid r \text{ is a good seed}\}$.*

If we come up with a good seed $r \in G$ then we can find some $x \in f^{-1}(1)$ just by running D-PAIRWISE-HITTER$(\varepsilon, \frac{1}{2}, n, r)$ — this costs $\frac{2}{\varepsilon}$ queries and no random bits. Thus we have a new problem: finding elements of G inside $\{0,1\}^{2n}$. For $f \in F(n, \varepsilon)$, PAIRWISE-HITTER$(\varepsilon, \frac{1}{2}, n, r)$ should succeed with probability at least $\frac{1}{2}$ (since we fixed its δ at $\frac{1}{2}$), so at least half the choices of r are good seeds: $|G| \geq \frac{1}{2}2^{2n}$. Hence the problem of finding good seeds is just the job for a hitter with $\varepsilon = \frac{1}{2}$. WALK-HITTER is an attractive choice, since it is optimal for such fixed ε. This yields the following algorithm.

Algorithm: COMPOSED-HITTER$^f(\varepsilon, \delta, n)$

Define the following deterministic subroutine (using the oracle f and the inputs ε and n):

$$\text{CHECK-SEED}(r) \equiv \left[\text{return } f\left(\text{D-PAIRWISE-HITTER}^f(\varepsilon, \tfrac{1}{2}, n, r)\right)\right] \qquad \text{where } r \in \{0,1\}^{2n}$$

Compute $r \leftarrow \text{WALK-HITTER}^{\text{CHECK-SEED}}(2n, \frac{1}{2}, \delta)$; that is, execute WALK-HITTER with input $(2n, \frac{1}{2}, \delta)$ and answer its oracle queries using CHECK-SEED.
Output D-PAIRWISE-HITTER$(\varepsilon, \frac{1}{2}, n, r)$.

Proposition 12.4.1 COMPOSED-HITTER *is a hitter.*

Proof: Note that by definition r is a good seed iff CHECK-SEED$(r) = 1$. Since $f \in F(n, \varepsilon)$ and PAIRWISE-HITTER is a sampler, at least half of the r values are good, i.e., CHECK-SEED $\in F(2n, \frac{1}{2})$. Thus by WALK-HITTER being a hitter:

$$\Pr\left[\text{WALK-HITTER}^{\text{CHECK-SEED}}(\tfrac{1}{2}, \delta, 2n) \in G\right] > 1 - \delta$$

Hence with probability at least $1 - \delta$, WALK-HITTER outputs a good seed r, and in this case D-PAIRWISE-HITTER$(\varepsilon, \frac{1}{2}, n, r)$ outputs x such that $f(x) = 1$. ■

The complexities of COMPOSED-HITTER are:

- $Q_{\text{COMPOSED-HITTER}}(\varepsilon, \delta, n) \le Q_{\text{WALK-HITTER}}(\frac{1}{2}, \delta, 2n) \cdot Q_{\text{PAIRWISE-HITTER}}(\varepsilon, \frac{1}{2}, n) = O\left(2 \cdot \log\left(\frac{1}{\delta}\right)\right) \cdot \left(\frac{2}{\varepsilon}\right) = O\left(\frac{1}{\varepsilon}\right) \cdot \log\left(\frac{1}{\delta}\right)$
- $R_{\text{COMPOSED-HITTER}}(\varepsilon, \delta, n) = R_{\text{WALK-HITTER}}(\frac{1}{2}, \delta, 2n) = 2n + O\left(\log\left(\frac{1}{\delta}\right)\right)$

Both complexities are optimal, since they match the corresponding lower bounds (up to a constant factor).

12.4.2 A Composed Sampler

We have obtained an optimal hitter by comining two suboptimal hitters. Can we do the same for samplers? There are some difficulties: first, we haven't seen any sampler with the characteristics of WALK-HITTER (namely, optimal query complexity for constant ε). Furthermore, in the two-level construction we have seen, the first level needs to somehow combine the results of the second level. For hitters this was trivial, but here it's not clear whether using a sampler in the first level to estimate the average of the second one would work. By appropriate adaptation of the two-level scheme we indeed obtain an optimal sampler.

As before, we use a construction based on pairwise-independent sample spaces for the second level. PAIRWISE-SAMPLER uses $2n$ bits of randomness, so we consider a deterministic version D-PAIRWISE-SAMPLER of it which accepts these $2n$ bits as an additional input. That is,

$$\text{PAIRWISE-SAMPLER}^f(\varepsilon, \delta, n) \equiv \left[r \in_R \{0,1\}^{2n} \text{ ; return D-PAIRWISE-SAMPLER}^f(\varepsilon, \delta, n, r)\right]$$

Definition 12.8 (Good Seed) $r \in \{0,1\}^{2n}$ *is called a* good seed *(w.r.t. f, ε, n) if* $\left|\text{D-PAIRWISE-SAMPLER}^f(\varepsilon, \frac{1}{100}, n, r) - \overline{f}\right| < \varepsilon$. *Denote* $G \stackrel{\text{def}}{=} \{r \mid r \textit{ is a good seed}\}$.

Unlike the case for hitters, we can't easily know whether a seed is good. However, we know that *most* seeds are good, and we can use random walk on an expander graph to obtain a set of seeds in which, with high probability, most seeds are good. This intuition yields the following algorithm.

Algorithm: COMPOSED-SAMPLER$^f(\varepsilon, \delta, n)$

Choose a strongly-constructible expander graph $G_{2^{2n}}$ with an algebraic expansion coefficient $\lambda \leq \frac{1}{100}$ and degree $d = \text{poly}\left(\frac{1}{\lambda}\right) = \frac{1}{O(1)}$ (the constants are fixed in the proof below).
Set $q = O\left(\log\left(\frac{1}{\delta}\right)\right)$
Conduct a random walk of length q on $G_{2^{2n}}$ to generate a sequence of q seeds $r_1, \ldots, r_q \in \{0,1\}^{2n}$.
For each $i \in \{1, \ldots, q\}$, compute $s_i =$ D-PAIRWISE-SAMPLER$(\varepsilon, \frac{1}{100}, n, r_i)$.
Output the median of $\{s_1, \ldots, s_q\}$.

To prove correctness we need the following generalization of Theorem 6 in Lecture 10:

Theorem 12.9 *Let $G = (V, E)$ be an expander with algebraic expansion coefficient λ. Let $B_1, \ldots B_l \subseteq V$. Let the random variables $v_1, \ldots, v_l$ denote the vertices encountered in a random walk of length l on G with a random starting point. For any $B \subseteq V$ denote $\rho(B) = \frac{|B|}{|V|}$. Then:*

$$\Pr\left[\forall i : v_i \in B_i\right] \leq \sqrt{\rho\left(B_1\right)} \prod_{i=2}^{l} \left(\sqrt{\rho\left(B_i\right)} + \lambda\right)$$

Proof: The proof of this theorem is very similar to the proof of Theorem 6 in Lecture 10. For a given set $B \subseteq V$, define the projection:

$$P_B(x) = \begin{cases} 1 & \text{if } x \in B \\ 0 & \text{otherwise} \end{cases}$$

We define π to be the vector in R^V for which $\forall v \in V : \pi_v = \frac{1}{n}$ (i.e., π represents the uniform distribution on V). The following holds (similarly to the proof of Theorem 6 in Lecture 10):

$$\Pr\left[\forall i : v_i \in B_i\right] = \left\| P_{B_l} A P_{B_{l-1}} A \ldots P_{B_1} \pi \right\|_2$$

Recall the following lemma:

Lemma 12.4.2 (Lemma 3.1 in Lecture 10) *For every set $B \subseteq V$ and vector $\alpha \in R^V$:*

$$\left\|P_B A \alpha\right\|_2 \leq \left(\sqrt{\rho\left(B\right)} + \lambda\right) \cdot \|\alpha\|_2$$

Denote $N = |V|$. By l successive applications of Lemma 12.4.2, we obtain:

$$\begin{aligned}
\|P_{B_l} A P_{B_{l-1}} A \ldots P_{B_1} \pi\|_1 &\leq \sqrt{N} \cdot \left\|P_{B_l} A P_{B_{l-1}} A \ldots P_{B_1} \pi\right\|_2 \\
&\leq \sqrt{N} \cdot \textstyle\prod_{i=2}^{l} \left(\sqrt{\rho(B_l)} + \lambda\right) \|P_{B_1} \pi\|_2 \\
&= \sqrt{N} \cdot \textstyle\prod_{i=2}^{l} \left(\sqrt{\rho(B_l)} + \lambda\right) \sqrt{\frac{\rho(B_1)}{N}} \\
&= \sqrt{\rho\left(B_1\right)} \textstyle\prod_{i=2}^{l} \left(\sqrt{\rho\left(B_i\right)} + \lambda\right)
\end{aligned}$$

■

Using Theorem 12.9, we prove the following:

Lemma 12.4.3 *Let $G = (V, E)$ be an expander with algebraic expansion coefficient λ. Let $B \subseteq V$ and denote $\rho(B) = \frac{|B|}{|V|}$. Let the random variables $v_1, \ldots, v_l$ denote the vertices encountered in a random walk of length l on G with a random starting point. Then:*

$$\Pr\left[|\{i : v_i \in B\}| \geq \tfrac{1}{2}l\right] \leq \left(8\sqrt{\rho(B)} + 8\lambda\right)^{\frac{1}{2}l}$$

Proof: For a given $I \subseteq \{1, \ldots, l\}$, we set:

$$B_i^I = \begin{cases} B & \text{if } i \in I \\ V & \text{otherwise} \end{cases}$$

Using Theorem 12.9:

$$\begin{aligned}
\Pr\left[|\{i : v_i \in B\}| \geq \tfrac{1}{2}l\right] &\leq \sum_{I \subseteq \{1,\ldots,l\},\, |I| \geq \frac{1}{2}l} \Pr\left[\forall i \in V : v_i \in B_i^I\right] \\
&< \sum_{I \subseteq \{1,\ldots,l\},\, |I| \geq \frac{1}{2}l} \sqrt{\rho(B_1^I)} \prod_{i=2}^{l} \left(\sqrt{\rho(B_i^I)} + \lambda\right) \\
&< \sum_{I \subseteq \{1,\ldots,l\},\, |I| \geq \frac{1}{2}l} \prod_{i=1}^{l} \left(\sqrt{\rho(B_i^I)} + \lambda\right) \\
&< \sum_{I \subseteq \{1,\ldots,l\},\, |I| \geq \frac{1}{2}l} \left(\prod_{i \in I} \left(\sqrt{\rho(B_i^I)} + \lambda\right) \cdot \prod_{i \notin I} \left(\sqrt{\rho(B_i^I)} + \lambda\right)\right) \\
&= \sum_{I \subseteq \{1,\ldots,l\},\, |I| \geq \frac{1}{2}l} \left(\prod_{i \in I} \left(\sqrt{\rho(B)} + \lambda\right) \cdot \prod_{i \notin I} (1 + \lambda)\right) \\
&= \sum_{I \subseteq \{1,\ldots,l\},\, |I| \geq \frac{1}{2}l} \left(\left(\sqrt{\rho(B)} + \lambda\right)^{|I|} \cdot \left(1 + \lambda\right)^{l-|I|}\right) \\
&\leq \sum_{I \subseteq \{1,\ldots,l\},\, |I| \geq \frac{1}{2}l} \left(\left(\sqrt{\rho(B)} + \lambda\right)^{\frac{1}{2}l} \cdot \left(1 + \lambda\right)^{\frac{1}{2}l}\right) \\
&\leq 2^l \cdot \left(1 + \lambda\right)^{\frac{1}{2}l} \cdot \left(\sqrt{\rho(B)} + \lambda\right)^{\frac{1}{2}l}
\end{aligned}$$

Since $\lambda < 1$, we obtain:

$$\begin{aligned}
\Pr\left[|\{i : v_i \in B\}| \leq \tfrac{1}{2}l\right] &< 2^{\frac{3}{2}l} \left(\sqrt{\rho(B)} + \lambda\right)^{\frac{1}{2}l} \\
&= \left(8\sqrt{\rho(B)} + 8\lambda\right)^{\frac{1}{2}l}
\end{aligned}$$

■

To prove correctness of our sampler we also need the following technical fact:

Fact 12.4.4 *Let $A = \{a_1, \ldots, a_n\}$ be a set of real numbers and $x < y$ be also real numbers. If there exists a subset $I \subseteq A$ s.t. $|I| > \frac{1}{2}n$ and all elements in I fall in the interval (a, b) then the median of A also falls in this interval. In other words, if $|\{a \in A : x < a < y\}| > \frac{1}{2}n$, then $x < \text{median}(A) < y$.*

Proof: Define:

$$G = \{a \in A \mid a \geq \text{median}(A)\}$$
$$L = \{a \in A \mid a \leq \text{median}(A)\}$$

By the definition of a median: $|G| \geq \lceil \frac{1}{2}n \rceil$ and $|L| \geq \lceil \frac{1}{2}n \rceil$. Since $|I| > \frac{1}{2}n$, there exist some $g \in G \cap I$ and $l \in L \cap I$. Because $g \in I$ we have $g < y$, so using $g \in G$ we get $\text{median}(A) \leq g < y$. Similarly, because $l \in I$ we have $x < l$, so using $l \in L$ we get $\text{median}(A) \geq l > x$. The claim follows. ■

We can now prove correctness of the sampler.

Proposition 12.4.5 Composed-Sampler *is a sampler.*

Proof: Consider the execution of Composed-Sampler$^f(\varepsilon, \delta, n)$ with some $f : \{0,1\}^n \to \{0,1\}$.

Recall that the algorithm produces q seeds $\{r_1, \ldots, 1_q\}$ and runs d-Pairwise-Sampler on each of them to obtain corresonding values $\{s_1, \ldots, r_q\}$. For each seed r_i, if r_i is a good seed then by definition: $|s_i - \bar{f}| < \varepsilon$. If strictly more than one half of $\{r_1, \ldots, r_q\}$ are good seeds then by Fact 12.4.4, the output $s = \text{median}\{s_1, \ldots, s_q\}$ also satisfies $|s - \bar{f}| < \varepsilon$. Thus, it suffices to show that with probability at least δ, strictly more than one half of $\{r_1, \ldots, r_q\}$ are good seeds.

For every ε, n and $f \in F(n)$, at least $\frac{99}{100}$ of the seeds are good (by Pairwise-Sampler$(\varepsilon, \frac{1}{100}, n)$ being a sampler), i.e., $|G| \geq \frac{99}{100}|\{0,1\}^{2n}|$. Letting $B = \{0,1\}^{2n} \setminus G$ be the set of "bad" seeds, we obtain $\rho(B) \leq \frac{1}{100}$. Thus, by invoking Lemma 12.4.3 on G_{2n} we obtain:

$$\begin{aligned}
\Pr\left[\text{more than one half of } \{r_1, \ldots, r_q\} \text{ are good seeds}\right] &= 1 - \Pr\left[|\{i : r_i \in B\}| > \tfrac{1}{2}q\right] \\
&\leq \left(8\sqrt{\rho(B)} + 8\lambda\right)^{\frac{1}{2}q} \\
&\leq \left(8 \cdot \sqrt{\frac{1}{100}} + 8 \cdot \frac{1}{100}\right)^{\frac{1}{2}q} \\
&\leq 0.88^{\frac{1}{2}q} = \delta
\end{aligned}$$

q is chosen as to satisfy the last inequality. ■

The complexities of Composed-Sampler are as follow (where $q = O\left(\log\left(\frac{1}{\delta}\right)\right)$ is the number of tested seeds):

- $Q_{\text{Composed-Sampler}}(\varepsilon, \delta, n) = q \cdot Q_{\text{Pairwise-Sampler}}(n, \varepsilon, \frac{1}{100})) = O\left(\log\left(\frac{1}{\delta}\right)\right) \cdot \frac{1}{\varepsilon^2 \cdot 100} = O\left(\log\left(\frac{1}{\delta}\right) \cdot \frac{1}{\varepsilon^2}\right)$

- $R_{\text{Composed-Sampler}}(\varepsilon, \delta, n) = 2n + O\left(\log\left(\frac{1}{\delta}\right) \cdot \log(d)\right) = 2n + O\left(\log\left(\frac{1}{\delta}\right)\right)$

This result is optimal, since it matches the lower bounds (up to a constant factor).

Bibliographic Notes

Our discussion of hitters and samplers is based on [20]. The median-of-averages sampler is due to Bellare, Goldreich and Goldwasser [11].

Lecture 13

Randomized Rounding

Notes taken by Ido Shaked, Ariel Elbaz and David Lehmann

Summary: Some approximation algorithms use linear programming on a variant of the original problem. The solutions of a linear programming problem are non-integral, in the general case, while legitimate solutions to the original problems are integers only. One idea which can be used to turn real numbers into integer solutions, is to use the linear programming (non-integer) solutions as probability parameters for selecting (integer) solutions to the original problem. In this lecture we present this idea and demonstrate it in two cases.

13.1 The General Idea

Given an instance of a boolean optimization problem (e.g. Max-SAT which is NP-Hard), we may reduce that problem to an analogue problem of Integer Linear Programming (ILP). Doing so, we arrive at an equivalent problem for which all the feasible solutions are integers in the range [0, 1] which means only 0,1, and such that every optimal solution for the ILP problem expresses an optimal solution for the original boolean problem. We then treat that problem as an instance of a simple Linear Programming, by dropping the integrality requirement, (note that this problem is known to be in P) and solve that problem, getting a solution expressed as Real numbers in the range [0,1]. We then use Random Rounding, a method we describe here, to convert the solution to a solution in integers 0,1 (which translates to a solution in T,F for the boolean problem), and which will have a good chance of becoming a "good" solution (one that is relatively "close" to the optimal solution).

13.2 Application to Max-SAT

Definition 13.1 Max-SAT:

Input: *A boolean formula* φ *over the variables* $\overrightarrow{X} = (x_1, ..., x_n)$ *with the following structure*

$$\varphi = \bigwedge_{j=1}^{m} C_j = \bigwedge_{j=1}^{m} (\bigvee_{i \in P_j} x_i) \vee (\bigvee_{i \in N_j} \neg x_i)$$

where N_j *and* P_j *are the sets of negated and unnagated variables in clause* C_j *respectively.*

Desired Output: *A truth assignment for $\overrightarrow{X}$ that satisfies the maximal number of clauses C_j.*

13.2.1 The Basic Method

We now translate an instance of Max-SAT to an analogue instance of Integer Linear Programming (ILP), as follows.

$$
\begin{array}{llll}
 & max\{\sum_{j=1}^{m} y_j\} & & \\
\text{st} & (i) \quad \forall j \in [m] & : & 0 \le y_j \le 1, \quad y_j \in \mathsf{Z} \\
 & (ii) \quad \forall i \in [n] & : & 0 \le x_i \le 1, \quad x_i \in \mathsf{Z} \\
 & (iii) \quad \forall j \in [m] & : & y_j \le \sum_{i \in P_j} x_i + \sum_{i \in N_j} (1 - x_i).
\end{array}
$$

Any optimal solution to the ILP problem is an optimal solution to the Max-SAT problem, and vice versa.

> Oded's Note: *Looking at the ILP formulation, Condition (ii) forces the x-variables to be Boolean (or have values in $\{0,1\}$), whereas the other conditions force a correspondance between the y_j's and indicators representing whether the corresponding clauses are satisfied by the assignment to the x_i's.*

We now remove the integrality requirement and arrive at the following Linear Programming (LP) problem:

$$
\begin{array}{llll}
 & max\{\sum_{j=1}^{m} y_j\} & & \\
\text{st} & (i) \quad \forall j \in [m] & : & 0 \le y_j \le 1 \\
 & (ii) \quad \forall i \in [n] & : & 0 \le x_i \le 1 \\
 & (iii) \quad \forall j \in [m] & : & y_j \le \sum_{i \in P_j} x_i + \sum_{i \in N_j} (1 - x_i).
\end{array}
$$

It is known that LP is solvable in polynomial time using, say, the Ellipsoid method. Thus:

Theorem 13.2 *LP is in P.*

Denote the optimal solution to the LP problem $\hat{X} = (\hat{x}_1, ..., \hat{x}_n), \hat{Y} = (\hat{y}_1, ..., \hat{y}_m)$.
Note that $\hat{x}_i$'s are in the range [0,1]. We will use those values as probabilities when rounding to integers 0 and 1. Suppose $\overrightarrow{Y^*} = (y_1^*, \ldots, y_n^*)$ is an optimal solution to the ILP. Clearly $\sum_{j=1}^{n} \hat{y}_j \ge \sum_{j=1}^{n} y_j^*$.

Randomized Rounding: We obtain a new integer solution by rounding: every x_i is determined by $x_i = \begin{cases} 1 & \textit{with probability } \hat{x}_i \\ 0 & \textit{otherwise} \end{cases}$.
This technique is called Randomized Rounding (R.R.).

Definition 13.3 *Size of Clause:* $c_j \stackrel{\text{def}}{=} |P_j| + |N_j|$.

Lemma 13.2.1 *Every clause C_j is satisfied, with probability at least $(1-\frac{1}{e})\cdot\hat{y}_j$, by the assignment obtained from the Randomized Rounding. That is:*

$$\begin{aligned}
\Pr[C_j \textit{ is satisfied by } R.R.(\hat{X})] &\geq (1-(1-\frac{1}{c_j})^{c_j})\cdot\hat{y}_j \\
&\geq (1-\frac{1}{e})\cdot\hat{y}_j
\end{aligned}$$

As a direct corollary to Lemma 13.2.1, we get:

Corollary 13.4 *The expected fraction of clauses that are satisfied by Randomized Rounding on an optimal LP solution is greater than 0.6.*

Proof: The expected number of clauses that are satisfied by Randomized Rounding is

$$\begin{aligned}
&\sum_{j=1}^{m}\Pr[C_j \text{ is satisfied by } R.R.(\hat{X})] \\
&\quad\geq \sum_{j=1}^{m}(1-\frac{1}{e})\cdot\hat{y}_j \\
&\quad= (1-\frac{1}{e})\cdot\sum_{j=1}^{m}\hat{y}_j \\
&\quad\geq (1-\frac{1}{e})\cdot\sum_{j=1}^{m}y_j^* \quad\approx\quad 0.6\cdot\sum_{j=1}^{m}y_j^*
\end{aligned}$$

■

Proof of Lemma 13.2.1: Fixing any $j \in [m]$, we have.

$$\begin{aligned}
&\Pr[C_j \text{ is not satisfied by } R.R.(\hat{X})] \\
&\quad= \prod_{i\in P_j}\Pr[x_i=0]\cdot\prod_{i\in N_j}\Pr[x_i=1] \\
&\quad= \prod_{i\in P_j}(1-\hat{x}_i)\cdot\prod_{i\in N_j}\hat{x}_i
\end{aligned}$$

Define, for every i in $P_j \cup N_j$, $\hat{z}_i = \begin{cases} 1-\hat{x}_i & i\in P_j \\ \hat{x}_i & i \in N_j \end{cases}$. By Condition (iii), we have

$$\hat{y}_j \leq \sum_{i\in P_j}\hat{x}_i + \sum_{i\in N_j}(1-\hat{x}_i) = \sum_{i\in P_j\cup N_j}(1-\hat{z}_i)$$

and hence

$$\sum_{i\in P_j\cup N_j}\hat{z}_i \leq c_j - \hat{y}_j.$$

Clearly

$$\begin{aligned}
&\prod_{i \in P_j} (1-\hat{x}_i) \cdot \prod_{i \in N_j} \hat{x}_i \\
&= \prod_{i \in P_j \cup N_j} \hat{z}_i \\
&\leq \max_{\substack{\overrightarrow{W} = (w_1, \ldots, w_n) \\ \sum_{i \in P_j \cup N_j} w_i \leq c_j - \hat{y}_j}} \left\{ \prod_{i \in P_j \cup N_j} w_i \right\}
\end{aligned}$$

Note that the product of m non-negative variables, when their sum is bounded, is maximized when they are all equal. Thus, since we have $\sum_{i \in P_j \cup N_j} w_i \leq c_j - \hat{y}_j$, we derive the maximum $\overrightarrow{W^*}$ to be when all w_i^* are equal; that is, $w_i^* = (\frac{c_j - \hat{y}_j}{c_j})$ for every $i \in P_j \cup N_j$. It follows that

$$\prod_{i \in P_j \cup N_j} w_i^* = (\frac{c_j - \hat{y}_j}{c_j})^{c_j} = (1 - \frac{\hat{y}_j}{c_j})^{c_j}$$

So far we have proven that for every clause C_j,

$$\Pr[C_j \text{ is satisfied by } R.R.(\hat{X})] \geq 1 - (1 - \frac{\hat{y}_j}{c_j})^{c_j}$$

To complete our proof we need to show that

$$1 - (1 - \frac{\hat{y}_j}{c_j})^{c_j} \quad \geq \quad (1 - (1 - \frac{1}{c_j})^{c_j}) \cdot \hat{y}_j \quad \geq \quad (1 - \frac{1}{e}) \cdot \hat{y}_j \tag{13.1}$$

We will use the equality : $1^c - b^c = (1-b) \cdot \sum_{i=1}^{c-1} b^i$. (*)

Specifically, applying (*) for $b = (1 - \frac{\hat{y}_j}{c_j})$ and $c = c_j$, we get:

$$\begin{aligned}
1 - (1 - \frac{\hat{y}_j}{c_j})^{c_j} &= (1 - (1 - \frac{\hat{y}_j}{c_j})) \cdot [\sum_{i=1}^{c_j - 1} (1 - \frac{\hat{y}_j}{c_j})^i] \\
&= \frac{\hat{y}_j}{c_j} \cdot [\sum_{i=1}^{c_j - 1} (1 - \frac{\hat{y}_j}{c_j})^i] \\
&\geq \frac{\hat{y}_j}{c_j} \cdot [\sum_{i=1}^{c_j - 1} (1 - \frac{1}{c_j})^i] \\
&= \frac{1}{c_j} \cdot [\sum_{i=1}^{c_j - 1} (1 - \frac{1}{c_j})^i] \cdot \hat{y}_j \\
&= (1 - (1 - \frac{1}{c_j})^{c_j}) \cdot \hat{y}_j
\end{aligned}$$

where the last equality is due to applying (*) for $b = (1 - \frac{1}{c_j})$ and $c = c_j$.

This proves the first inequality in Eq. (13.1). The second inequality (i.e., $(1 - (1 - \frac{1}{c_j})^{c_j}) \cdot \hat{y}_j \geq (1 - \frac{1}{e}) \cdot \hat{y}_j$) is derived from the fact that, for every $x \geq 1$, it holds that $(1 - \frac{1}{x})^x < \frac{1}{e}$. ■

13.2.2 Improving the Max-SAT approximation algorithm

The above approximation algorithm achieves an approximation ratio of $(1-\frac{1}{e})$ of the optimal solution. Yet there is a known approximation algorithm with an approximation ratio of $\frac{3}{4}$. We wish to modify the algorithm we have just seen, to achieve an approximation ratio of $\frac{3}{4}$.

Consider a random assignment for all the variables, i.e. $\overrightarrow{X} \in_R \{T,F\}^n$. Each clause C_j is not satisfied if and only if every literal in it has a false value. If we assign each variable X_i, independently, a random value from $\{T,F\}$, the probability that C_j is not satisfied is 2^{-c_j} (c_j, as before, is the number of literals in C_j), hence the probability that C_j is satisfied is $(1-2^{-c_j})$.

We note that for any j, when using a random assignment, the probability of C_j being satisfied increases with the size of the clause, c_j. However, using the assignment we got from solving the linear programming problem, the probability of C_j being satisfied decreases with c_j. Noting this difference calls for some combination of the two methods, to get a method that gives higher probability for each clause being satisfied regardless of it's size.

A $\frac{3}{4}$-approximation algorithm: Given a boolean formula $\varphi = \bigwedge_{j=1}^{m} C_j$ as above, we first run the previous algorithm, using Linear Programming and Randomized Rounding, and get a solution $\overrightarrow{X^1}$. We also select a random assignment $\overrightarrow{X^2} \in_R \{T,F\}^n$. We output $\overrightarrow{X^1}$ with probability $\frac{1}{2}$, $\overrightarrow{X^2}$ with probability $\frac{1}{2}$.

Analysis We claim that the expected number of satisfied clauses is at least $\frac{3}{4}$ of the optimum.

We define a new random variable v_j^1, which, for each clause C_j, will be 1 if the clause is satisfied using the assignment $\overrightarrow{X^1}$, or 0 otherwise. We use v_j^1's to write an expression for the expected number of satisfied clauses, which is simply the sum of these variables $E(\sum_{j=1}^{m} v_j^1)$, which is equal to $\sum_{j=1}^{m} E(v_j^1)$. The expected number of satisfied clauses, when using $\overrightarrow{X^1}$, is at least $\sum_{j=1}^{m}(1-(1-\frac{1}{c_j})^{c_j})\cdot \hat{y}_j$, as shown above.

We also define the variables v_j^2's, for use with the assignment $\overrightarrow{X^2}$. Using $\overrightarrow{X^2}$, we've seen that for every j $E(v_j^2) = (1-2^{-c_j})$.

Taking each of the two assignments with probability $\frac{1}{2}$, the expected number of satisfied clauses is

$$\begin{aligned}
E(\sum_{j=1}^{m} v_j) &= \frac{1}{2}\cdot E\left(\sum_{j=1}^{m} v_j^1\right) + \frac{1}{2}\cdot E\left(\sum_{j=1}^{m} v_j^2\right) \\
&\geq \frac{1}{2}\cdot \sum_{j=1}^{m}(1-(1-\frac{1}{c_j})^{c_j})\cdot \hat{y}_j + \frac{1}{2}\cdot \sum_{j=1}^{m}(1-2^{-c_j}) \\
&\geq \sum_{j=1}^{m}\left(\frac{1-(1-\frac{1}{c_j})^{c_j}}{2} + \frac{1-2^{-c_j}}{2}\right)\cdot \hat{y}_j \\
&\geq \sum_{j=1}^{m} \frac{3}{4}\cdot \hat{y}_j \\
&= \frac{3}{4}\cdot \sum_{j=1}^{m} \hat{y}_j
\end{aligned}$$

where the last inequality uses the following claim:

Claim 13.2.2 *For any positive integer x it holds that $\frac{1}{2}(1-(1-\frac{1}{x})^x)+\frac{1}{2}(1-2^{-x}) \geq \frac{3}{4}$.*

Proof: It is sufficient to prove that for any positive integer x it holds that $(1-\frac{1}{x})^x + 2^{-x} \leq \frac{1}{2}$. The claim can easily be checked for $x = 1$ and 2.

> Oded's Note: *For $x = 1$ we have $(1-\frac{1}{x})^x + 2^{-x} = 0 + 0.5 = 0.5$, whereas for $x = 2$ we have $(1-\frac{1}{x})^x + 2^{-x} = 0.5^2 + 2^{-2} = 0.5$.*

For any $x \geq 3$, we have

$$(1-\frac{1}{x})^x + 2^{-x} < e^{-1} + 2^{-3} < \frac{1}{2}$$

■

13.3 Application to the Minimal-Set-Cover problem

Definition 13.5 Minimal Set Cover: *Given n and $\{S_j\}_{j=1}^m$, a collection of sets which cover $[n]$, a* **minimal set cover** *is a minimal sub-collection, J, which still covers $[n]$; that is, a minimal J for which $\bigcup_J S_j = [n]$.*

13.3.1 The approximation algorithm

We will go by the same path as we did for the MaxSAT approximation algorithm: first we translate the problem to an ILP problem, then solve the related LP problem, and finally use Randomized Rounding to get an Integer solution from the (non Integer) solution of the LP problem.

For every i, we name C_i to be the collection of sets that contain i. Now we rephrase the min-set-cover problem: to cover every i, we must take into J at least one of the subsets S_j that contain i, and these are all in C_i. We also should find a minimal such J. We can write the rephrased problem as an Integer Linear Programming problem:

$$\begin{array}{llcll} & min\{\sum_{j\in[m]} y_j\} & & & \\ \text{st} \quad (i) & \forall j \in [m] & : & 0 \leq y_j \leq 1 & , y_j \in \mathsf{Z} \\ \quad (ii) & \forall i \in [n] & : & \sum_{j\in C_i} y_j \geq 1 & \end{array}$$

From a solution to this system corresponds to a cover $J = \{S_j \mid y_j = 1\}$. Requirement (ii) ensures that J indeed covers every i, and the minimal sum ensures the minimality of J.

We now remove the integrality requirement from the y_j's, and get a LP problem. We denote by *Opt* the optimal solution of the LP problem. That is, denoting the optimal LP solution by $\hat{y}_j$'s, we have $Opt \stackrel{\text{def}}{=} \sum_{j\in[m]} \hat{y}_j$.

Applying Randomized Rounding. The $\hat{y}_j$'s can be seen as a measure of the importance of S_j in the cover. However, should we, as previously in the Max-SAT problem, choose J simply by taking each S_j with probability $\hat{y}_j$, we will end up with J that has a small expected size (i.e., *Opt* which is at most as big as the optimum of the ILP), but will probably leave some of the i's uncovered. For example, with $C_i = \{1,2\}$ and a possible solution $\hat{y}_1 = \hat{y}_2 = 0.5$, with probabilty 1/4 none of these sets is taken (and the element i remains uncovered).

To ensure that no element remains uncover, we will take a larger random cover. Specifically, we will include S_j in J with probability $\min\{1,\ K \cdot \hat{y}_j\}$ (rather than with probability $\hat{y}_j$), where $K \stackrel{\text{def}}{=} 2\ln(2n)$.

13.3.2 Analysis

Claim 13.3.1 *The expected size of this cover is, at most, $K \cdot Opt$.*

Proof: $E(|J|) = \sum_{j=1}^{m} \Pr(j \in J) \leq \sum_{j=1}^{m} K \cdot \hat{y}_j = K \cdot Opt.$ ■

Claim 13.3.2 *For every i, $\Pr(i \text{ isn't covered}) \leq \frac{1}{2n}$.*

Proof: If, for any j in C_i, $\hat{y}_j$ is at least $\frac{1}{K}$, then S_j is always in J (because $K \cdot \hat{y}_j \geq 1$) and i is always covered. We therefore focus on the case where $\hat{y}_j < \frac{1}{K}$.

Let us define, for every j in C_i, a new random variable:

$$\eta_j = \begin{cases} 1 & \textit{with probability } K \cdot \hat{y}_j \\ 0 & \textit{otherwise} \end{cases}$$

The above is well-define because $K \cdot \hat{y}_j < 1$. Furthermore, since $K \cdot \hat{y}_j = \min(K \cdot \hat{y}_j, 1)$, the random variable η_j indicates whether or not j was taken to J. Thus, for every i in $[n]$:

$$\Pr(i \text{ isn't covered}) = \Pr\left(\sum_{j \in C_i} \eta_j = 0\right). \tag{13.2}$$

We will use the following Chernoff bound

A Chernoff bound: *For independent 0-1 random variables $x_1, ..., x_n$, with $\lambda = E(\sum_{i=1}^{n} x_i)$ and any $\epsilon \in [0,1]$, it holds that*

$$\Pr\left[\sum_{i=1}^{n} x_i \leq (1-\epsilon)\lambda\right] \leq e^{\frac{-\epsilon^2\lambda}{2}}$$

In our case, we have $\lambda \stackrel{\text{def}}{=} E(\sum_{j \in C_i} \eta_j) = K \cdot \sum_{j \in C_i} \hat{y}_j$, which by Condition (ii) is at least K. Applying the above bound (with $\epsilon = 1$), we get

$$\begin{aligned} \Pr[\sum_{j \in C_i} \eta_j = 0] &= \Pr[\sum_{j \in C_i} \eta_j \leq 0] \\ &\leq e^{-\lambda/2} \leq e^{-K/2} \end{aligned}$$

Recalling that $K = 2\ln(2n)$, we have $e^{-K/2} = 1/2n$ and the claim follows. ■

Conclusion: We have shown that the probability that any specific i is not covered is at most $\frac{1}{2n}$. Using the union bound, it follows that the probability that there is an uncovered $i \in [n]$ is at most $\frac{1}{2}$. Thus, J is a collection of subsets S_j, with expected size no more than $2\ln(2n) \cdot Opt$, which, with probability at least $\frac{1}{2}$, covers $[n]$. Furthermore, with constant probability, J has size $O(\log n) \cdot Opt$ and covers all $[n]$.

Bibliographic Notes

The *Randomized Rounding* technique was introduced by Raghavan and Thompson [39]. The application to MaxSAT is due Goemans and Williamson [19].

Bibliography

[1] L.M. Adleman and M. Huang. *Primality Testing and Abelian Varieties Over Finite Fields.* Springer-Verlag Lecture Notes in Computer Science (Vol. 1512), 1992. Preliminary version in *19th ACM Symposium on the Theory of Computing*, 1987.

[2] L.M. Adleman, K. Manders and G. Miller. On Taking Roots in Finite Fields. In *18th IEEE Symposium on Foundations of Computer Science*, pages 175–177, 1977

[3] M. Ajtai, J. Komlos, E. Szemerédi. Deterministic Simulation in LogSpace. In *19th ACM Symposium on the Theory of Computing*, pages 132–140, 1987.

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

[5] N. Alon. Eigenvalues and expanders. *Combinatorica*, Vol. 6, pages 83–96, 1986.

[6] N. Alon, L. Babai and A. Itai. A fast and Simple Randomized Algorithm for the Maximal Independent Set Problem. *J. of Algorithms*, Vol. 7, pages 567–583, 1986.

[7] N. Alon, J. Bruck, J. Naor, M. Naor and R. Roth. Construction of Asymptotically Good, Low-Rate Error-Correcting Codes through Pseudo-Random Graphs. *IEEE Transactions on Information Theory*, Vol. 38, pages 509–516, 1992.

[8] N. Alon, O. Goldreich, J. Håstad, R. Peralta. Simple Constructions of Almost k-wise Independent Random Variables. *Journal of Random structures and Algorithms*, Vol. 3, No. 3, (1992), pages 289–304.

[9] N. Alon and V.D. Milman. λ_1, Isoperimetric Inequalities for Graphs and Superconcentrators, *J. Combinatorial Theory, Ser. B*, Vol. 38, pages 73–88, 1985.

[10] N. Alon and J.H. Spencer. *The Probabilistic Method*, John Wiley & Sons, Inc., 1992.

[11] M. Bellare, O. Goldreich, and S. Goldwasser. Randomness in Interactive Proofs. *Computational Complexity*, Vol. 4, No. 4, pages 319–354, 1993.

[12] M. Bellare, O. Goldreich, and E. Petrank. Uniform Generation of NP-witnesses using an NP-oracle. *Inform. and Comp.*, Vol. 163, pages 510–526, 2000.

[13] C.H. Bennett, G. Brassard and J.M. Robert. Privacy Amplification by Public Discussion. *SIAM Journal on Computing*, Vol. 17, pages 210–229, 1988.

[14] R. Canetti, G. Even and O. Goldreich. Lower Bounds for Sampling Algorithms for Estimating the Average. *Information Processing Letters*, Vol. 53, pages 17–25, 1995.

[15] L. Carter and M. Wegman. Universal Hash Functions. *Journal of Computer and System Science*, Vol. 18, 1979, pages 143–154.

[16] B. Chor and O. Goldreich. On the Power of Two–Point Based Sampling. *Jour. of Complexity*, Vol 5, 1989, pages 96–106. Preliminary version dates 1985.

[17] A. Cohen and A. Wigderson. Dispensers, Deterministic Amplification, and Weak Random Sources. In *30th IEEE Symposium on Foundations of Computer Science*, 1989, pages 14–19.

[18] O. Gaber and Z. Galil. Explicit Constructions of Linear Size Superconcentrators. *Journal of Computer and System Science*, Vol. 22, pages 407–420, 1981.

[19] M. Goemans and D. Williamson. New 3/4-approximation algorithms for the maximum satisfiablity problem. *SIAM Journal on Discrete Mathematics*, Vol. 7, No. 4, pages 656–666, 1994.

[20] O. Goldreich. A Sample of Samplers – A Computational Perspective on Sampling. *ECCC*, TR97-020, May 1997.

[21] O. Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness. Algorithms and Combinatorics series* (*Vol. 17*), *Springer*, 1999.

[22] O. Goldreich, R. Impagliazzo, L.A. Levin, R. Venkatesan, and D. Zuckerman. Security Preserving Amplification of Hardness. In *31st IEEE Symposium on Foundations of Computer Science*, pages 318–326, 1990.

[23] O. Goldreich and A. Wigderson. Tiny Families of Functions with Random Properties: A Quality–Size Trade–off for Hashing. *Journal of Random structures and Algorithms*, Vol. 11, Nr. 4, December 1997, pages 315–343.

[24] S. Goldwasser and J. Kilian. Primality Testing Using Elliptic Curves. *Journal of the ACM*, Vol. 46, pages 450–472, 1999. Preliminary version in *18th ACM Symposium on the Theory of Computing*, 1986.

[25] J. Håstad, S. Phillips and S. Safra. A Well Characterized Approximation Problem. *Information Processing Letters*, Vol. 47:6, pages 301–305. 1993.

[26] R. Impagliazzo, L.A. Levin and M. Luby. Pseudorandom Generation from One-Way Functions. In *21st ACM Symposium on the Theory of Computing*, pages 12–24, 1989.

[27] R. Impagliazzo and D. Zuckerman. How to Recycle Random Bits. In *30th IEEE Symposium on Foundations of Computer Science*, 1989, pages 248–253.

[28] M. Jerrum, L. Valiant and V. Vazirani. Random Generation of Combinatorial Structures from a Uniform Distribution. *Theoretical Computer Science*, Vol. 43, pp. 169–188, 1986.

[29] A. Joffe. On a set of almost deterministic k-independent random variables. *Annals of Probability*, Vol. 2 (1), pages 161–162, 1974.

[30] N. Kahale, Eigenvalues and Expansion of Regular Graphs. *Journal of the ACM*, 42(5):1091–1106, September 1995.

[31] A. Lubotzky, R. Phillips, P. Sarnak, Ramanujan Graphs. *Combinatorica*, Vol. 8, pages 261–277, 1988.

[32] F.J. MacWilliams and N.J.A. Sloane. *The Theory of Error-Correcting Codes*. North-Holland, Amsterdam, The Netherlands, 1977.

[33] G.A. Margulis. Explicit Construction of Concentrators. *Prob. Per. Infor.* 9 (4) (1973), 71–80. (In Russian, English translation in *Problems of Infor. Trans.* (1975), 325–332.)

[34] R. Motwani and P. Raghavan. *Randomized Algorithms*, Cambridge University Press, 1995.

[35] J. Naor and M. Naor. Small-bias Probability Spaces: Efficient Constructions and Applications. *SIAM J. on Computing*, Vol 22, 1993, pages 838–856.

[36] V. Pratt. Every Prime has a Succinct Certificate. *SIAM Journal on Computing*, Vol. 4, pages 214–220, 1975.

[37] M.O. Rabin. Digitalized Signatures and Public Key Functions as Intractable as Factoring. MIT/LCS/TR-212, 1979.

[38] M.O. Rabin. Probabilistic Algorithm for Testing Primality. *Journal of Number Theory*, Vol. 12, pages 128–138, 1980.

[39] P. Raghavan and C.D. Thompson. Randomized Rounding. *Combinatorica*, Vol. 7, pages 365–374, 1987.

[40] M. Sipser. A Complexity Theoretic Approach to Randomness. In *15th ACM Symposium on the Theory of Computing*, pages 330–335, 1983.

[41] R. Solovay and V. Strassen. A Fast Monte-Carlo Test for Primality. *SIAM Journal on Computing*, Vol. 6, pages 84–85, 1977. Addendum in *SIAM Journal on Computing*, Vol. 7, page 118, 1978.

[42] L. Stockmeyer. On Approximation Algorithms for #P. *SIAM Journal on Computing*, Vol. 14 (4), pages 849–861, 1985. Preliminary version in *15th ACM Symposium on the Theory of Computing*, pages 118–126, 1983.

[43] U.V. Vazirani. Randomness, Adversaries and Computation. Ph.D. Thesis, EECS, UC Berkeley, 1986.

[44] A. C. Yao. Probabilistic computations: Towards a Unified Measure of Complexity. In *17th FOCS*, pages 222–227, 1977.