Course on Pseudorandomness

(Lecture Notes)

JAYALAL SARMA

Department of Computer Science and Engineering
Indian Institute of Technology Madras (IITM)
Chennai, India

Last updated on : April 30, 2021

Table of Contents

Todo list

Part I

General Introduction and Tools

Power of Randomization and Derandomization Problem

Lecturer : Jayalal Sarma

WEEK 1

Randomized algorithms play a very powerful role in algorithm design. We will concentrate on the randomized algorithms for decision problems in this course. So all of our computational problems can be abstractly represented as given a string $x \in \Sigma^*$ in an alphabet, does x have property $\mathcal{P}$ or not?

Informally, a randomized algorithm running in time t is an algorithm that that on input x is allowed to perform at most t instances of random experiment of tossing unbiased coins during its computation and uses the outcome of the experiment in the computation, but however, provides a guarantee that the answer of the algorithm is the *correct* answer for the input x in a good fraction of the possible outcomes of the experiment. [1]

1.1 Randomness helps in Matrix Multiplication Verification

Consider the task of multiplying two $n \times n$ matrices over the field $\mathbb{F}_2$. The trivial algorithmic solution to the problem takes $O(n^3)$ time and the trivial lower bound for the problem is $\Omega(n^2)$. It has been a long standing question which is the right complexity bound for this important problem (improvement to which will lead to improvements even in practice !). The exponent of matrix multiplication is the smallest constant ω such that two $n \times n$ matrices may be multiplied by performing $O(n^{\omega+\epsilon})$ for every $\epsilon > 0$.

Indeed, one of the basic ideas that we learn in algorithms courses for demonstrating the power of divide and conquer is the Strassen's multiplication which gives a running time bound of $O(n^{2.,73})$ which went through a sequence of improvements and to the current best of $O(n^{2.31})$.

The question that we address now is something closely related - that of verifying whether a given multiplicaiton is correct.

PROBLEM 1.1.1. *Given three matrices $A, B, C \in \mathbb{F}_2^{n \times n}$, check whether $AB = C$ or not.*

Indeed, the trivial method would be to mutliply the two matrices and check if the result is equal to C. But then this requires, $O(n^{2.31} + n^2)$ time using the best matrix multiplication algorithm that we know currently. Since we just require verification, it is conceivable that we might be able to do better if we are allowed to make a small some error in the process. We show that this indeed possible to be done in $O(n^2)$ with error probability at most 2^{-k} for any constant k (independent of

[1] Notice that if the guarantee for the algorithm is not saying "*strictly more than half fraction of the coin tosses*", then essentially the algorithm is useless since we can always replace it with a random experiment of tossing an unbiased coin and returning the answer to be YES if we get the heads and NO if we get tails. Note that we have at least a $\frac{1}{2}$ probability of success $\frac{1}{2}$.

n).

Trivial Approach using randomization: A natural first cut attempt is to choose an entry $(i, j) \in [n] \times [n]$ uniformly at random from the n^2 entries of C and checking if :

$$\sum_{k=1}^{n} A_{ik} B_{kj} = C_{ij}$$

This runs in time $O(n)$. And we can choose constant k more entries to amplify the success probability. However, the probability of correctness is very small. Suppose, in the worst case input, there was only one $(i, j) \in n \times n$ where there was an error in the multiplication. The probability that we will choose that particular (i, j) for verification is as small as $\frac{1}{n^2}$. Amplifying this to a success probability of $\frac{1}{2^k}$ takes more than $\Omega(n^{1+\epsilon})$ iterations and hence the overall algorithm will take $\Omega(n^{2+\epsilon})$ time which is beyond what we can afford to spend time on.

Freivalds' Approach: The idea is to check a randomly chosen "linear combination" of entries rather than a single entry of the matrix C. If we choose this to be a random linear combination of rows of the matrix C, then the combinatorics helps to achieve a much better probability of error. We now formally write down the algorithm and analyse it.

Algorithm 1.1 : Frievald's Algorithm for Verification of Matrix Multiplication - $\mathcal{F}(A, B, C)$

1: Choose a vector $r \in \mathbb{F}_2^n$ uniformly at random.
2: If $[A(Br) = Cr]$ then output YES else output NO.

The computation of $(A(Br))$ and Cr are done using $O(n^2)$ time algorithms since computing a linear transformation result Ax for an $n \times n$ matrix can be done in $O(n^2)$ time. Now we argue correctness guarantees. If the given matrices indeed satisfy $AB = C$, then no matter which $r \in \mathbb{F}_2^n$ algorithm chooses in step 1, $ABr = Cr$ and hence it always will output YES. The error can happen only when $AB \neq C$ and the algorithm ends up choosing an unfortunate r such that $ABr = Cr$. The following claim upper bounds the probability of this .

CLAIM 1.1.2. *For any $A, B, C \in \mathbb{F}_2^n$ such that $AB \neq C$,*

$$\Pr[\mathcal{F}(A, B, C) \textit{ outputs } \text{YES}\,] \leq \frac{1}{2}$$

Proof. Let $A, B, C \in \mathbb{F}_2^n$ such that $AB \neq C$. We need to analyze the probability that $ABr = Cr$ for $r \in \mathbb{F}_2^n$ chosen uniformly at random. If $AB \neq C$, then $D = AB - C$ is a non-zero matrix. Thus

$$\Pr_{r \in \mathbb{F}_2^n}[ABr \neq Cr] = \Pr_{r \in \mathbb{F}_2^n}[Dr \neq 0]$$

Imagine that D was all 1s matrix. Now the above probability is exactly the number of vectors $r \in \mathbb{F}_2^n$ with an odd number of 1s in it. By an obvious bijection, this is also the number of subsets of $[n]$ with odd size. The latter is exactly 2^{n-1} and hence the probability of such a vector r being chosen from $\mathbb{F}_2^n$ is $\frac{1}{2}$.

Now we formalize and generalize this. Let p be the vector Dr. Since $D \neq 0$, there must be an entry $D_{ij} \neq 0$. Define, $A = \{j : D_{ij} \neq 0\}$. We know that $A \neq \phi$. $i, j \in [n]$. Thus :

$$p_i = \sum_{k=1}^{n} D_{ik} r_k = \sum_{k \in A} r_k$$

$$\text{Note that: } \Pr_{r \in \mathbb{F}_2^n}[Dr = 0] \leq \Pr_{r \in \mathbb{F}_2^n}[p_i = 0]$$

Notice that the latter probability depends only on r_k where $k \in A$. The fraction of assignments of the bits $\{r_k : k \in A\}$ which makes $p_i = 0$ is exactly $\frac{1}{2}$ since the number of even sized subsets of A and number of odd sized subsets of A are exactly the same. □

REMARK 1.1.3. Informally, if we run the above algorithm $\mathcal{F}(A, B, C)$, and the algorithm outputs NO, then we can trust the answer and conclude that indeed $AB \neq C$. But a YES answer from the algorithm cannot be trusted - it could be because of the unfortunate choice of $r \in \mathbb{F}_2^n$ that came as the outcome of the experiment.

Why are we interested in the above algorithm even though it gives a success probability bound of only $\frac{1}{2}$? The reason is that, it is a one-sided error algorithm and hence still much better than a coin toss outcome because the algorithm does not make an error when $AB = C$. In fact, such algorithms can be repeated in a natural way - run k times, and if any of them says $AB \neq C$ output NO. This reduces the error probability in exponentially in k - since each of the trials should give an error (with probability $\frac{1}{2}$) and hence the error probability bound is at most $\frac{1}{2^k}$.

1.2 Polynomial Identity Testing Problem

The previous example, while it demonstrates the point, might be a bit unsatifactory since the problem under consideration anyway has an efficient algorithm. To address this, we will now see another example problem where there is an efficient (polynomial time in the input size) randomized algorithm for solving the problem but a deterministic algorithm for solving the problem is not known.

The problem is easy-to-state algorithmic question on polynomials. Fix $\mathbb{F}$ to be the field where the coefficients are chosen from. *Given a polynomial $p \in \mathbb{F}[x_1, x_2, \ldots x_n]$, test if it is identically zero.* That is, do all the terms cancel out and become the zero polynomial.

This problem has its roots in the simple high school arithmetic. Suppose we are given a polynomial in a complicated form where the monomials may repeat with arbitrary coefficients etc. We want to find out if the coefficient of the monomials cancel out to zero. This in effect is testing whether the polynomial is the zero polynomial, and equivalenty it is testing if the polynomials evaluates to zero on all substitutions of the variable from the underlying field $\mathbb{F}$.

How are we given the polynomial? This indeed is going to have effect on the complexity of the problem. Let us start with the high school arithmetic again. Suppose we are given it in the monomial form (though some monomials may repeat) along with their coefficients. To solve the problem, it suffices to check, for each monomial whether the coefficient in its various appearences is adding up to zero. Given the explicit representation at the input, this is very easy to do by simply going over the input for each monomial. Hence this can be done in time polynomial in the

input.

What if the polynomial is not given that explicity. How can it be given implicitly compared to the list of monomials? One answer is that, we could give it in a bracketed form. That is, the polynomial $x_1x_2 + x_1x_4 + x_3x_2 + x_3x_4$ can be given as $(x_1 + x_3)(x_1 + x_4)$. Indeed, this is implicit, since an expression of length n, can have number of actual number of monomials to be 2^n - consider the example $(x_1 + x_2)(x_3 + x_4)\dots(x_{n-1} + x_n)$ where n is the number of variables.

What is the most implicit form that we can think of? A black box which evaluates the polynomial. That is, we have an oracle p when given input a returns $p(a)$, the value of polynomial at a.

Assume that we are also given an upper bound on the degree of the polynomial $deg(p) \leq d$. Indeed, we do not have access to the actual polynomial except through the blackbox. We have to use some property of the degree d polynomials. The most obvious one is the number of points in which they can evaluate to zero. Based on this thought, the following deterministic algorithm solves the problem.

Algorithm 1.2 A deterministic algorithm for univariate polynomial identity testing

1: Choose $d + 1$ different points $a_1, \dots, a_{d+1}$.
2: Call the oracle $d + 1$ times to evaluate $p(a_1), \dots, p(a_{d+1})$.
3: If all calls returned 0 accept else reject.

If p were really the zero polynomial then all calls will return 0 and we will definitely accept. If p were not 0, then at most d calls can return 0 since a polynomial with degree at most d has at most d roots. Hence if $p \neq 0$, then our algorithm will definitely reject.

Multivariate PIT: Now let us think about the problem when p is a multivariate polynomial. The previous assertion that a degree d polynomial has at most d roots no longer holds. To see this, consider the degree 2 polynomial $p(x_1, x_2) = x_1x_2$. This has an infinite number of roots $x_1 = 0, x_2 \in \mathbb{F}$, where $\mathbb{F}$ is the (possibly infinite) field over which p is defined.

We can work around this problem by considering a finite subset of the field, say $S = \{0, \dots, 10\}$. The polynomial p has 19 zeroes. So if x_1, x_2 is chosen uniformly at random from S there is at most $19/100$ chance that we will get a false result. As can be seen from the above example, by making the size of S arbitrarily large, we can make the error probability arbitrarily small. But then the disadvantage is that we will need more random bits in order to choose an element at random from the set $|S|$, and the running time of our algorithm will also increase.

Generalizing this strategy that we will follow is as follows: If the total degree of the polynomial is $\leq d$, and if $S \subseteq \mathbb{F}$, such that $|S| \geq 2d$, instead of picking elements arbitrarily, we pick elements uniformly at random from S. Indeed, there may be many choices for the values which may lead to zero. But how many?

LEMMA 1.2.1 (**Schwartz-Zippel Lemma**). *Let $p(x_1, x_2, \cdots, x_n)$ be a non-zero polynomial over a field $\mathbb{F}$. Let $S \subseteq \mathbb{F}$*

$$\Pr_{\bar{a} \in S^n}[p(\bar{a}) = 0] \leq \frac{d}{|S|}$$

Proof. (By induction on n) For $n = 1$: For a univariate polynomial p of degree d, there are $\leq d$

roots. Now in the worst case the set S that we picked has all d roots. Thus for a random choice of substitution for the variable from S, the probability that it is a zero of the polynomial p is at most $\frac{d}{|S|}$.

For $n > 1$, write the polynomial p as a univariate polynomial in x_1 with coefficients as polynomials in the variables $p(x_2, \ldots, x_n)$.

$$\sum_{j=0}^{d} x_1^j p_j(x_2, x_3, \ldots, x_n)$$

For example: $x_1x_2^2 + x_1^2x_2x_3 + x_3^2 = (x_2x_3)x_1^2 + (x_2^2)x_1 + x_1^0(x_3^2)$.

We need to analyze the probability that we will choose a zero of the polynomial (even though the polynomial is not identically zero). For a choice of the variables as $(a_1, a_2, \cdots, a_n) \in S^n$, we ask the question : how can $p(a_1, a_2, \cdots, a_n)$ be zero? It could be because of two reasons:

1. $\forall j \ : \ 1 \leq j \leq n, \ \ p_j(a_2, a_3, \ldots, a_n) = 0.$
2. Some coefficients $p_j(a_2, a_3, \ldots, a_n) = 0$ are non-zero, but the resulting univariate polynomial in x_1 evaluates to zero upon substituting $x_1 = a_1$.

Now we are ready to calculate $Pr[p(a_1, a_2, \ldots, a_n) = 0]$. For a random choice of $(a_1, \ldots, a_n)$. Let A denote the event that the polynomial $p(a_1, \ldots, a_n) = 0$. Let B denote the event that $\forall j \ : \ 1 \leq j \leq n, \ \ p_j(a_2, a_3, \cdots, a_n) = 0$. Note that, $Pr[A] = Pr[A \wedge B] + Pr[A \wedge \overline{B}]$.

We calculate both the terms separately: $Pr[A \wedge B] = Pr[B].Pr[A|B] = Pr[B]$ where the last equality is because $B \Rightarrow A$. Let ℓ be the highest power of x_1 in $p(x)$. That is $p_\ell \neq 0$. Since the event B insists that for all j, $p_j(a_2, a_3, \ldots, a_n) = 0$, we have that $Pr[B] \leq Pr[p_\ell(a_2, a_3, \ldots, a_n) \neq 0]$. By induction hypothesis, since this polynomial has only $n-1$ variables and has degree at most $\frac{d-\ell}{S}$. Thus, $Pr[B] \leq \frac{d-\ell}{S}$.
To calculate the other term,

$$Pr[A \cap \overline{B}] \quad = \quad Pr[\overline{B}].Pr[A|\overline{B}] \leq Pr[A|\overline{B}] \leq \frac{\ell}{|S|}$$

where the last inequality holds because the degree of the non-zero univariate polynomial after substituting for $a_2, \ldots, a_n$ is at most ℓ and hence the base case applies. □

This suggests the following efficient algorithm for solving PIT. Given d and a blackbox evaluating the polynomial p of degree at most d.

Algorithm 1.3 : Schwartz-Zippel Algorithm for Multivariate PIT

1: Choose $S \subseteq \mathbb{F}$ of size $\geq 4d$.
2: Choose $(a_1, a_2, \ldots, a_n) \in_R S^n$.
3: Evaluate $p(a_1, a_2, \ldots a_n)$ by querying the blackbox.
4: If it evaluates to 0 accept else reject.

The algorithm runs in time $\mathsf{poly}(n)$. The following Lemma states the error probability and follows from the Schwartz-Zippel Lemma that we saw before.

LEMMA 1.2.2. *There is a randomized polynomial time algorithm A, which, given a black box access to a polynomial p of degree d (d is also given in unary), answers whether the polynomial is identically zero or not, correctly with probability at least $\frac{3}{4}$.*

Notice that in fact the lemma is weak in the sense that it ignores the fact that when the polynomial is identically zero then the success probability of the algorithm is actually 1 !. In other words, is it is a one-sided error randomized algorithm.

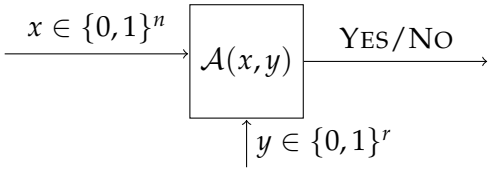
1.3 Derandomization Problem

In the previous section (lecture), we talked about randomized algorithms for problems for which we do not know deterministic algorithms with similar complexity resource bounds. Indeed, we are not happy about randomized algorithms as such since these algorithms require perfect unbiased coin toss experiments to be performed and we do not have them in practice. Indeed, the fact that they can output erroneous answers, even though with low probability makes them useless in critical practical applications.

How do convert them to deterministic algorithms without causing much overhead?. One possible way is to look at each algorithm and use inherent properties of the problem to analyze the randomized algorithm better to come up with ways to remove randomness from that algorithm. Here, we start with the original randomized algorithm for a particular problem, and improve it to derandomize it and the techniques are usually very algorithm specific. We will do some examples of this kind later in the course.

1.3.1 Abstract Model of Derandomziation

From now on, we will be concentrating only on abstract models of these randomized algorithms. We fix some notations first. A randomized algorithm $\mathcal{A}$ on input x runs in time $t(n)$ (where $n = |x|$) and let $y \in \{0,1\}^{r(n)}$ be the concatenation of the unbiased coin toss experiement that the algorithm does during its execution. Notice that $r(n) \leq t(n)$ (we drop the n when it is not required explicitly). If the algorithm runs in polynomial time $t(n) \leq n^c$ for a constant c independent of n.



The guarantee we have is there is an $\epsilon \in (0, \frac{1}{2}]$.

$$\forall x \in \{0,1\}^n, \Pr_{y \in \{0,1\}^r}[A(x,y) \text{ is correct.}] \geq \frac{1}{2} + \epsilon$$

1.3.2 Derandomization by Brute Force Approach

The trivial approach to obtain an equivalent deterministic algorithm is run over all possible outcomes of the experiment and check the answer from the algorithm for each of them. Whichever answer comes as majority - report that as the final answer.

Algorithm 1.4 ($\mathcal{A}'$) : input $x \in \{0,1\}^n$, where success prob. $\frac{1}{2} + \epsilon$ for $\mathcal{A}$

1: $count \leftarrow 0$.
2: **for** `each` $y \in \{0,1\}^r$ **do**
3: `Check if` $\mathcal{A}(x,y)$ `accepts, if so increment` *count*
4: **end for**
5: If $[count > 2^{r-1}]$ then output YES else output NO.

If the running time of the randomized algorithm $\mathcal{A}$ is $t(n)$, then the running time of the new algorithm (which is deterministic) is $t(n)2^{r(n)}$. To argue correctness, if the actual answer for input $x \in \{0,1\}^n$ is YES, then the fraction of $y \in \{0,1\}^r$ which makes $\mathcal{A}(x,y)$ accept is strictly more than $\frac{1}{2}$ and hence the algorithm will output YES. If the actual answer for input $x \in \{0,1\}^n$ is NO, then then the fraction of $y \in \{0,1\}^r$ which makes $\mathcal{A}(x,y)$ accept is strictly less than $\frac{1}{2}$ and hence the algorithm will output NO.

REMARK 1.3.1. Note that the algorithm $\mathcal{A}$ will run in $\mathsf{poly}(n)$ time if the original randomized algorithm was running in $t \leq \mathsf{poly}(n)$ time and was using $r \leq O(\log n)$ random bits.

1.4 Pseudorandomness : An Informal Overview

Ideally, we would like to replace the randomized algorithm with a deterministic one as done in the previous section. However, we know how to do this trivially only when the randomized algorithm uses $O(\log n)$ random bits.

We outline two "out of the box" thoughts related to our target of derandomization of randomized algorithms.

Fooling the Algorithm with Pseudorandom bits : PRGs The first one is about using $y \in \{0,1\}^r$ as not independent random bits. But use *dependent* random bits instead. Indeed, the analysis for the error bound for the algorithm $\mathcal{A}$ now may fail since it may assume total independence between the bits of y in its mathematical argument. However, sometimes, it is possible that same analysis (or even a better analysis) may work even when the bits of the y are dependent in a limited way [2] But this may be specific to the algorithm and sometimes to the problem itself. We would ideally want a more abstract strategy which would work for randomized algorithms in general, modelled by what we described in the previous section. But even if we made it work with some dependent randombits, how do we produce this distribtion of y' with the desired limited dependence among them? Construction of the methods which can produced limited dependence thus becomes important.

Taking a more abstract view point, informally, we would like to have a box (formally an algorithm G) which takes in pure random bit string of length $y' \in \{0,1\}^{r'}$ and produces a string $y \in \{0,1\}^r$ such that the distribution of y "looks" pseudo-random for the resource limited algorithm $\mathcal{A}$. The idea is that we will run the algorithm $\mathcal{A}$ with the random string provided the $G(y')$

[2] At one extreme, if we had an algorithm and an analysis which works wen all the bits of the y are the same (which is an example of extreme dependence) we dont require the random bit at all - we can directly simulate the algorithm deterministically the trivial way.

- the output of G on input $y' \in \{0,1\}^{r'}$ chosen uniformly at random.

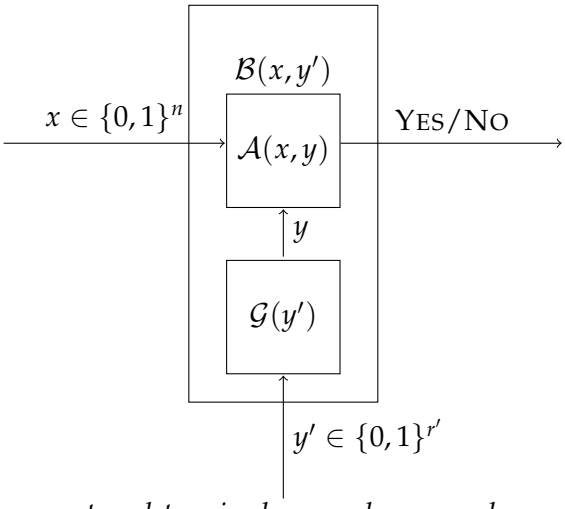
Indeed, since we are not providing pure random bits to $\mathcal{A}$. Hence we should expect its correctness guarantees to not hold good anymore. That is, it will deteriorate a bit. Can we guarantee that it does not deteriorate too much? This can be done in two ways (1) rework or reanalyse the algorithm $\mathcal{A}$ and argue that it is still having success probability greater than $\frac{1}{2}$ (which is enough for trivial derandomization) (2) use only resource bounds of $\mathcal{A}$ to argue that the change of y to $G(y')$ will not affect the success probability much. For this, the generator has to satisfy certain properties.

A function $G : \{0.1\}^{r'} \to \{0,1\}^{r}$ is said to be a Peudo-Random Generator (PRG) for complexity measure[3] s and error parameter $\delta \in [0,1]$ if, for any algorithm $\mathcal{A}$ which runs in time $t \leq s$ (or having complexity measure bounded by s): For any x,

$$\left| \Pr_{y \in \{0,1\}^r}[\mathcal{A}(x,y) \text{ Accepts}] - \Pr_{y' \in \{0,1\}^{r'}}[\mathcal{A}(x, G(y')) \text{ Accepts}] \right| \leq \delta$$

Connecting to the informal description, δ is the quantity by which the success probability deteriorates because of the use of the pseudorandom generator output, instead of pure random bits. Hence if we ensure that $\delta < \epsilon$, even after the use of the pseudorandom generator output, we still will have a randomized algorithm $\mathcal{B}$ with the following guarantee $\forall x \in \{0,1\}^n$:

$$\Pr_{y \in \{0,1\}^r}[\mathcal{B}(x,y) \text{ is correct.}] \geq \frac{1}{2} + \epsilon - \delta > \frac{1}{2}$$



We will end the description by asking the question - *what parameters determine how good our pseudrandom generator is?*. As per the above discussion it is:

- The relative values of r and r'. This leads to the definition of the *stretch* of the psuedorandom generator. We would ideally want an exponential stretch function so that with $r' \in O(\log n)$ we can produce y for $\mathcal{A}$ which is of length $r = O(n^c)$ for constant c.
- The value of s. This determines how powerful an algorithm can the pseudorandom generator manage to fool. The larger the s the better. Ideally we want s to be covering all polynomial time running time bounds.
- The value of δ. This determines the quantity by which the success probability of the algorithm deteriorates after plugging in the output of the pseudorandom generator instead of the y from the pure random bits. Ideally, we want $\epsilon - \delta > 0$. The smaller the δ, the better.

[3]We will make it more precise when it comes to the section where we handles these objects. We are leaving at the above description at a less precise level.

- Running time of the generator itself. Notice that we need $\mathcal{G}$ to be explicit polynomial time algorithm, which runs in time $\mathsf{poly}(n)$. That is, if $r' \in O(\log n)$ (which is what ideally we would want, so that the trivial derandomization runs in $\mathsf{poly}(n)$ time), then technically, the generator can run for exponential time in terms of its input size[4].

The main part of the game is in describing the generator algorithms (or functions from $\{0,1\}^{r'} \to \{0,1\}^{r}$). However, it is not even clear whether such functions exists for the range of parameters that we care about. Indeed, this is the kind of flavour that we will have.

- We can prove that the functions that we are looking for exist, with a non-constructive argument. This is done by - what is termed as the *Probablistic method*.
- Explicit descriptions of the functions, which are are required for the algorithms with required runtime bounds for $\mathcal{G}$ are not known. In fact, if we have such descriptions, then a complete derandomization of all randomized algorithms is possible, which will be a big achievement.

Refining Randomness : Randomness Extractors - Here is a completely different idea about supplying dependent randomness. We do not have source of pure random bits to supply for the randomized algorithm $\mathcal{A}$. But we may have impure radom bit sources. An imaginative question is *can we invest a few pure random bits in order to purify/extract and hence improve the impurity in the given random source?*. This, at first sounds crazy and leads to the following questions.

- How do we define *impure random bit sources*. They define distributions which are not uniform. There is the notion of entropy which can tell us how uniform the source is.
- How do we define *how good the output is*. Again, one could have used entropy here too naturally. However, noticing the fact that we would like to finally apply it our algorithms like $\mathcal{A}$, a different measure of "purity" is used which is the notion of statistical distance[5] to uniform distribution.

The above discussion leads to the definiton of a randomness extractor, which is a function $\mathcal{E} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ such that when X is a distribution on $\{0,1\}^n$ with entropy at least k, then the distribution of the output $\mathcal{E}(X, U_d)$ is ϵ-close to U_m where U_d and U_m are uniform distributions on the set $\{0,1\}^d$ and $\{0,1\}^m$ respectively.

Again, how do we determine how good is our extractor function? We want extractors which works on highly biased distributions (the smallest k possible) using fewest number of pure random bits (the smallest d possible) and produces output distributions which are closest to uniform distributions (ϵ must be smallest) - and still run in time polynomial in n.

Similar to pseudorandom generator functions, it is unclear apriori whether such functions even exist for the range or parameters we care about. A similar situation arises, where by using probablistic method, we can prove that such objects (functions) exists, but at the same time, we do not know how to construct them deterministically (equivalently describe the algorithm for $\mathcal{E}$).

[4] This marks the difference between the pseudorandom generators studied in cryptography and derandomization.

[5] Informally, this is the sum of the difference (in absolute value) between the probability values assigned to points in the sample space.

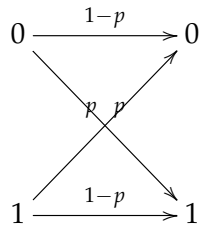
REMARK 1.4.1 ((Informal) - **Psuedo-random Objects**). The common flavour that we observed about the previous sections is that there are mathematical objects which we would like to describe (by providing algorithms to compute those functions) and we do know that the function that we seek exists. This situation is a common phenomenon in many objects. In fact, in most situations, it is not just the existance of the objects that is argued, but also that if the object that is of interest is chosen at random (with appropriately set up experiments), the object of interest shows up at the outcome with high probability. Thus, there is a randomized algorithm to explicitly construct the object, and now we have to derandomize them !. However, notice that in such situations, we can think of deranodomizations which just depends on those algorithms which chooses the object at random.

Other Contexts and Psuedorandom Objects We now describe a totally unrelated context in which the required mathematical functions display such a psuedorandom behaviour and an explicit construction is being sought for. The context is that of coding theory.

Coding theory had its inception in the late 1940's with the theory of reliable communication over a channel in the presence of noise - an area that started with the pioneering work of Claude Shannon and Richard Hamming. The former addressed and answered the fundamental questions about the possibility of the use of codes for reliable communication and the later develped some basic combinatorial constructions of error correcting codes that laid the foundations for the work later.

Theoretical computer scientists have a major role to play in the algorithmic aspects of coding theory research, and coding theory has proved to be instrumental in several interesting results in theoretical computer science as well. There has been several surprising applications of codes and the associated mathematical objects, in areas like algorithms, complexity theory and cryptography. Some part of the course will aim to discuss of those applications. However, we do not intend to be exhaustive.

The channel is not harmless in the real world. It introduces errors in the transmission. Depending on the application the error may be in the physical storage media (communication over time) or in the physical channel (communication over space). Some of the 0s gets flipped to 1s and vice versa, and some bits may get dropped too. For the purposes of this course we will study only model (Shannon studied several interesting variants), namely what are called Binary Symmetric Channels. In this model, each bit gets flipped with a probability p. That is, a 1 gets flipped to a 0 with probability and 0 gets flipped to 1 with probability p.



What is the natural strategy to cope up with errors in transmission? Create redundancy. For example, if Alice wants to send a bit 0 to Bob, she will do it five times, and send 11111 and ask Bob to take the majority of the bits as the bit that was sent. In this simple looking example we

have all the essence. The string that was sent will be called the *codeword* and the original bit to be sent is called the *message*. There are only two codewords 00000 and 11111 in the above example. If we define the notion of distance as the hamming distance, then the majority decoding mechanism described above can also be seen as choosing the codeword that is closest to the received word. This natural strategy of decoding is called *nearest neighbor decoding* or *maximum likelyhood decoding*.

Now let us observe facts about guarantees. Clearly if the channel is such that it will not corrupt more than 2 bits in a sequence of 5 bits, then Bob will be able to decode the message bit correctly. But the channel may actually flip more number of bits but with relatively lower probability. Thus if we increase the number of copies we make of the original message, with high probability (over the errors) introduced by the channel we are going to be able to decode the bit correctly.

To fix some notations, we denote $E : \{0,1\}^k \to \{0,1\}^n$ as the encoding function where k is the message length (in general) and n is the length of the codeword (which we will call the *block length*. Let $m \in \{0,1\}^k$ be a message, and $E(m) \in \{0,1\}^n$ is the transmitted word. The channel corrupts the message and let $y \in \{0,1\}^n$ is the received word. The error introduced by the channel could also be thought of as a string $\eta \in \{0,1\}^n$ where the η_i determines whether $y_i = (E(m))_i$ or not.

We want the following guarantee for any $m \in \{0,1\}^k$ as translating the above intuition:

$$Pr_\eta(D(E(m) + \eta) = m) \geq 1 - o(1)$$

where the $o(1)$ term is exponentially small depending on n and hence on k (since c is a constant).

Although the above statement is written in terms of a probability over choice of the channel error vector, a natural combinatorial guarantee that we would want is an encoding and decoding scheme such that if the error string η has weight at most $t < \frac{d}{2}$ the decoder retrieves the message correctly. That is, the encoder-decoder pair is guaranteed to get the message across the channel, if the number of corrputions by the chaneel is limited a number t. Indeed, the relative redundant information we sent should be minimised (which is the ration of k and n called the rate of the code).

Shannons theorem essentially states that under sutiable choice of the parameters there is a pair of encoding-decoding functions that can achieve this high confidence decoding of the original message. We will state the theorem formally only later. But again, the spirit of the theorem is that there does exist good encoding and decoding schemes with respect to the parameters we usually care about (which we make precise later). The area of algorithmic coding theory essentially attempts to address the question of constructing coding schemes for which there is an efficient decoding.

We conclude the lecture by stating that the three mathematical objects that we stated in this lecture do have some interconnections among themselves and also the pseudorandom objects that we are going to state in the next lecture too.

Method of Conditional Expectation, Pessimistic Estimators

Lecturer : Jayalal Sarma

WEEK

2

In this week, the plan is to learn the technique of conditional expection which is a derandomization technique that works for several algorithms. However, this is a technique which is specific to algorithms and not to problems. And there is no hard and fast rule by which we can say whether this technique is applicable for an algorithm. We demonstrate it with the MAXCUT problem.

2.1 MAXCUT Problem and Randomized Approximation

For an undirected graph $G(V, E)$, a cut is a partition of vertices into two sets $S, T \subseteq V$. The size of a cut is the number of edges that go across these partitions. That is, the size of the set :

$$cut(S, T) = \{e = (u, v) \mid (u, v) \in E, u \in S, v \in T\}$$

Maximum cut is a cut whose size is at least the size of any other cut. That is $|cut(S, T)|$ is the largest possible. Given a graph, the problem of finding a maximum cut in a graph is known as the MAXCUT problem.

The problem is hard: The MAXCUT problem is known to be NP-hard. This implies, in particular, that if we have an efficient algorithm for the MAXCUT problem, then some of the very hard problems will yield to having efficient algorithms solving them. This is believed to be unlikely.

Approximation algorithms: Hence it makes sense to talk about algorithms which may not output the exact maximum cut, but instead another cut. Indeed, this is useless unless there are guaranteed how large is the cut output by the algorithm. An example guarantee that we may want to target is, for the algorithm, no matter what the input graph G is, the cut output by the algorithm will be, say, at least $\left(\frac{1}{10}\right)$-th of the size of the maximum cut. This is called a 0.1-approximation algorithm[6]. Even with this relaxed target for the algorithm, is not immediately clear how to design such an algorithm. It turns out that the best known algorithm for MAXCUT does much better than this and achieves an approximation guarantee of 0.875, and for many reasons this is believed to be the best possible ratio that any polynomial time algorithm can achieve for MAXCUT problem. However, in this lecture, we will concentrate on much smaller rations.

Randomized Approximation algorithms: We resort to randomized algorithms - which in this context will be called randomized approximation algorithms. Notice that unlike the previous

[6]Exercise: if you have not seen it already, think about what would be a similar statement that you would like to target for a minimization problem, like vertex cover problem

examples, MAXCUT is not a decision problem. Hence, we need to be careful about designing and analysing randomized algorithms for it. For example, there is nothing like the algorithm being correct. Instead, we have only the notion of the approximation ratio - that is how close the output of the algorithm is, to the optimal value.

2.1.1 A Simple Randomized Algorithm

We start with a simple randomized algorithm for MAXCUT problem.

Algorithm 2.5 : Randomized Approx. Algorithm for MAXCUT for graph $G(V, E)$, $|V| = n$

$S = T = \phi$
for each $i \in [n]$ **do**
    Choose bit $b_i \in \{0, 1\}$ uniformly at random.
    if $b_i = 1$ **then**
        $S = S \cup \{i\}$
    else
        $T = T \cup \{i\}$.
    end if
end for
Output $cut(S, T)$.

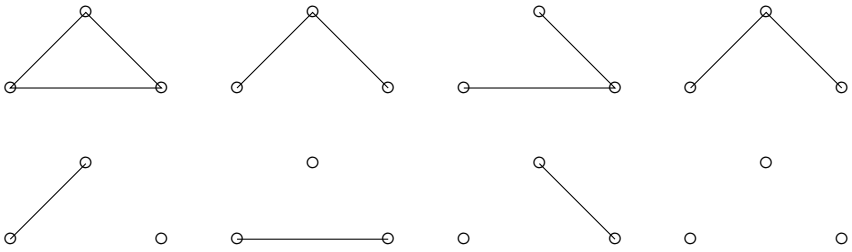
Notice that the sets S and T will form a partition of V at the end of the algorithm. Clearly, the algorithm will run in linear time. Indeed, the above description is also equivalent to - choose a random subset S of vertices from V and outputing $cut(S, \overline{S})$. Indeed, we need to give a guarantee about the size of the cut output by the algorithm. Roughly, the claim is that the average size of the cut (where average is taken over the 2^n different outcomes of the random choices). We recap some basics of random variables and expections now before stating the correctness claim.

2.1.2 Recap of Probability Basics, Random Variables, Expectation

Fix a set Ω, which is called the *sample space*. A probability distribution is defined over Ω, is a function $\Pr : \Omega \to [0, 1]$ satisfying the additional condition that, $\sum_{w \in \Omega} \Pr(w) = 1$. An event is a subset $\mathcal{E} \subseteq \Omega$. The probability of an even $\mathcal{E}$ is nothing buy the sum of the probability values by assigned by the distribution functon to the elements in the subset $\mathcal{E}$. That is, $\Pr(\mathcal{E}) = \sum_{w \in E} \Pr(w)$.

We consider an example which are going to be relevant for us. This is the notion of random graphs. Consider n vertices, and there are $\binom{n}{2}$ possible edges. Imagine that we will choose (independently) each edge to be present in our graph with probability p and to be absent in our graph with probability. The outcome of the experiment is an n-vertex simple graph and hence the sample space is the set of all n vertex graphs.

As an example, we can consider, $n = 3$. There are 3 possible edges and hence 8 possible graphs. The probability assigned the triangle graph (which is the complete graph on 3 vertices) is p^3 since all the three edges have to be chosen for this particular outcome to happen. In a similar way, the following pictures denote the sample space in this case with the corresponding probability values.



The probabilities in that order are p^3, $p^2(1-p)$, $p^2(1-p)$, $p^2(1-p)$, $p(1-p)^2$, $p(1-p)^2$, $p(1-p)^2$, $(1-p)^3$.

How do we analyse the probability that we get a connected graph as the outcome of the experiment. This is where events are used. Recall that formally, an event $\mathcal{E}$ is a subset of Ω.

$$Pr(\mathcal{E}) = \sum_{w \in \mathcal{E}} Pr(w)$$

In the above example, if the event $\mathcal{E}$ represent the set of connected graphs.

$$Pr(\mathcal{E}) = p^3 + 2p^2(1-p)$$

In the above example, if the event $\mathcal{E}'$ represent the set of bipartite graphs.

$$Pr(\mathcal{E}') = 1 - p^3$$

PROPOSITION 2.1.1 (**Subadditivity of Probability - a.k.a - Union theorem**). *Let $\mathcal{E}_1$,$\mathcal{E}_2 \ldots \mathcal{E}_n$ be events, then :*

$$Pr\left[\bigcup_i \mathcal{E}_i\right] \leq \sum_{i=1}^{n} Pr[\mathcal{E}_i]$$

DEFINITION 2.1.2 (**Conditional Probability**). *For two events $\mathcal{E}$ and $\mathcal{E}'$, we define,*

$$Pr(\mathcal{E}|\mathcal{E}') = \frac{Pr(\mathcal{E} \cap \mathcal{E}')}{Pr(\mathcal{E}')}$$

The conditional probability captures the questions of the kind, what is the probability that we get a connected graph if we are given that the outcome is a bipartite graph?

DEFINITION 2.1.3 (**Independent events**). *Two events $\mathcal{E}$ and $\mathcal{E}'$ are said to be independent, if*

$$Pr(\mathcal{E}|\mathcal{E}') = Pr(\mathcal{E})$$

Equivalently,

$$Pr(\mathcal{E} \cap \mathcal{E}') = Pr(\mathcal{E})Pr(\mathcal{E}')$$

For example, if we consider the events event $\mathcal{E}$ represent the set of connected graphs and event $\mathcal{E}'$ represent the set of bipartite graphs, then:

$$Pr(\mathcal{E} \cap \mathcal{E}') = 3p^2(1-p)$$

$$Pr(\mathcal{E})Pr(\mathcal{E}') = \left[(1-p)^3 + 3p(1-p)^2 + 3p^2(1-p)\right](1-p^3)$$

Since they are not equal, we conclude that the two events are not indendent. That is, the event that the graph is bipartite has an "influence" on the event that the graph is connected. To make this clearer, we suggest the following exercise:

Exercise 2.1.4. Let $G \in G(n,p)$. For all $S \subseteq V$, let A_S be the event that S forms an independent set in G. Show that if S and T are two distinct subsets of k vertices then A_S and A_T are independent if and only if $|S \cap T| \leq 1$.

Now, we will generalize the above notion of independence to more than two events. An event $\mathcal{E}$ is independent of a set of events $\{\mathcal{E}_j \mid j \in J\}$ if, for all subset $J' \subseteq J$, $Pr[\mathcal{E}| \cap_{j\in J} B_j] = Pr(A)$.

Exercise 2.1.5. Prove that an event $\mathcal{E}$ is independent of a set of events $\{\mathcal{E}_j \mid j \in J\}$ if and only if for all $J_1, J_2 \subseteq J$ such that $J_1 \cap J_2 = \phi$

$$\Pr[\mathcal{E} \cap \left(\cap_{j\in J_1} B_j\right) \cap \left(\cap_{j\in J_2} \overline{B_j}\right)] = \Pr(\mathcal{E}) \Pr[\left(\cap_{j\in J_1} B_j\right) \cap \left(\cap_{j\in J_2} \overline{B_j}\right)]$$

Let $\{\mathcal{E}_i \mid i \in I\}$ be a (finite) set of events. They are *pairwise independent* if for all $i \neq j$ the events $\mathcal{E}_i$ and $\mathcal{E}_j$ are independent. Events are *mutually independent* if each of them is independent from the set of the others. It is important to note that events may be pairwise independent but not mutually independent. Following exercise demonstrates that.

Exercise 2.1.6 (See Problem Set 1(Problem 1))**.** A random k-colouring fo a graph G is an element of the probability space (Ω, Pr) where Ω is the set of all k-colourings (i.e. partition of V into k sets $(V_1, V_2, \ldots, V_k)$, all this colourings being equally likely (so happening with probability $\frac{1}{k^n}$. For every edge e of G, let A_e be the event that the two endvertices of e receive the same colour. Show that:

(a) for any two edges e and f of G, the events A_e and A_f are independent.

(b) if e, f and g are three edges of a triangle of G, the events A_e, A_f and A_g are dependents.

Random Variables: We need the idea of random variables which we recap now. A random variable is another function $X : \Omega \to \mathbb{R}$. The expected value of the random variable is the "weighted average" value that it takes over the real numbers - weighted by the correspoding probability values. That is,

$$E[X] = \sum_{\alpha \in \mathbb{R}} \alpha \Pr[X = \alpha]$$

Indeed, $[X = \alpha]$ represents an event $\{w \in \Omega \mid X(w) = \alpha\} \subseteq \Omega$. Hence, the expectation can also be written equivalently as follows:

$$E[X] = \sum_{\alpha \in \mathbb{R}} \alpha \left(\sum_{\substack{w\in\Omega \\ X(w)=\alpha}} \Pr(w) \right) = \sum_{w\in\Omega} X(w) \Pr(w)$$

We need the following properties of expectation:

Tool 1 : Boolean Random Variables - Suppose X is a random variable that takes only Boolean values. In this case, $E[X] = \Pr[X = 1]$ which follows from the definitions.

Tool 2 : Linearity of Expectation : Suppose X_1 and X_2 are random variables defined based on the same probability distribution, consider the new random variable defined as $X = c_1X_1 + c_2X_2$. This is also a random variable as it is a function from $\Omega \to \mathbb{R}$ defined as $X(w) = c_1X_1(w) + c_2X_2(w)$ for every $w \in \Omega$. It turns out there is a neat relationship between the expectation of the random variables X, X_1 and X_2. This is one of the most imporant relation that is extensively used in analysis of randomized algorithms.

$$\begin{aligned} E[X] &= \sum_{w\in\Omega} X(w)\Pr(w) = \sum_{w\in\Omega} (c_1X_1(w) + c_2X_2(2))\Pr(w) \\ &= c_1\left(\sum_{w\in\Omega} X_1(w)\Pr(w)\right) + c_2\left(\sum_{w\in\Omega} X_2(w)\Pr(w)\right) = c_1E[X_1] + c_2E[X_2] \end{aligned}$$

We suggest practicing the application of linearity of expectation using the following exercise.

Exercise 2.1.7 (See Problem Set 1(Problem 2))**.** A graph $G = (V, E)$ is created at random by selecting each edge with probability p. What is the expected number of spanning trees in the randomly sampled graph? (*Hint : Use Cayley's Theorem that the number of distinct spanning trees on n vertices is n^{n-2}. Order them, and define an indicator random variable.*)

Tool 3 : Averaging Principle - Suppose X is a random variable and $E[X] = \mu$, then the following statements follow:

$$\exists w \in \Omega : X(w) \geq \mu \qquad \exists w \in \Omega : X(w) \leq \mu$$

Both of them can be proved by contradiction. For sample, we spell out the first one, suppose that the first statement is false. That is, $\forall w \in \Omega, X(w) < \mu$, then:

$$E[X] = \sum_{w\in\Omega} X(w)\Pr(w) < \sum_{w\in\Omega} (\mu \times \Pr(w)) = \mu\left(\sum_{w\in\Omega} \Pr(w)\right) = \mu$$

This implies, $E[X] < \mu$ which is a contrdiction. A similar proof holds for the other claim as well.

Tool 4 : Tail inequalities - Suppose we have a random variable X such that $E[X] = \mu$. What kind of probability guarantees can we write for X? For example, can we bound (in terms of the expectation) the probability that $X > \alpha$ for some $\alpha \in \mathbb{R}$? This is what tail bounds do. They help us write probability upper bounds based on expectations and other related parameters.

As a first example, consider a random variable that takes only non-negative values. Then we can write :

$$\textbf{Markov's Inequality :}\ \Pr[X \geq a] \leq \frac{E[X]}{a}$$

The proof is also quite simple.

$$E[X] = \sum_{\alpha \in \mathbb{R}} \alpha \Pr[X = \alpha] \geq \sum_{\alpha \geq a} \alpha \Pr[X = \alpha] \geq \sum_{\alpha \geq a} a \Pr[X = \alpha] \geq a \Pr[X \geq a]$$

For example, this helps us make statements of the form :

In particular, Markov's inequality implies the following bound on the probability that any random variable X is significantly larger than its expectation:

$$\Pr[X \geq (1+\delta)\mathbb{E}[X]] \leq \frac{1}{1+\delta} \tag{2.1}$$

For example, the probability that the random variable takes a value which is more than 4 times the expected value is at most 0.25. Unfortunately, this does not help us write down a probability bound for X taking value less than say $\frac{\mathbb{E}[X]}{4}$. Indeed, another form is :

Indeed, Markov's inequality is also pretty weak - as demonstrated by the following example - consider tossing n coins and X be the number of heads. Clearly $E[X] = \frac{n}{2}$. By Markov's inequality, $\Pr[X \geq n] \leq \frac{1}{2}$ but we know that it is much smaller than that, namely $\frac{1}{2^n}$.

What do we do if we want to bound the probability that the random variable takes a much lower value than the expectation? This is where we require more the next tail bound (without any assumption of positivity on the random variable).

Chebychev's Inequality : $\Pr[|X - \mu| \geq a] \leq \frac{Var[X]}{a^2}$ where, $Var[X] = \mathbb{E}[X^2] - \mathbb{E}[X]^2$

Proof of Chebychev's Inequality in the sum of random variables case: More often, we require the Chebychev's inequality when X is a random variable that is expressible as a sum of several independent random variables. We will handle that case. In particular, we will prove that:

$$\Pr\left[(X - \mu)^2 \geq a\right] \leq \frac{E[X]}{a} \tag{2.2}$$

$X = \sum_{i=1}^{k} X_i$. Let $\mathbb{E}[X_i] = \mu_i$ and $\mathbb{E}[X] = \sum_i \mu_i = \mu$ (say).

Define $Y_i = X_i - \mu_i$. By linearity of expectation, $\mathbb{E}[Y_i] = 0$. And let $Y = \sum_i Y_i = X - \mu$. The idea is to apply Markov's inequality to a postive random variable that can be formed out of Y. Indeed Y can take negative values, hence we can consider Y^2. We care about $\mathbb{E}[Y^2]$ in order to apply the Markov's inequality.

$$\mathbb{E}[Y^2] = \mathbb{E}\left[\left(\sum_i Y_i\right)\left(\sum_i Y_i\right)\right] = \mathbb{E}\left[\sum_{i,j} Y_i Y_j\right] = \sum_{i,j} \mathbb{E}[Y_i Y_j] = \sum_i \mathbb{E}[Y_i^2] + \sum_{i \neq j} \mathbb{E}[Y_i Y_j]$$

Now we will apply independence of the random variables, and conclude that $\mathbb{E}[Y_i Y_j] = \mathbb{E}[Y_i]\mathbb{E}[Y_j]$ but then in our case $\mathbb{E}[Y_i] = 0$. Using this:

$$\mathbb{E}[Y^2] = \sum_i \mathbb{E}[Y_i^2] = \sum_i \left[\mu_i(1-\mu_i)^2 + (1-\mu_i)\mu_i^2\right] = \sum_i \mu_i(1-\mu_i) < \mu$$

Hence, the theorem (equation 2.2) follows from Markov's inequality. The general form also has a similar proof. And note that $Var[Y] = \mathbb{E}[Y^2]$ since $\mathbb{E}[Y] = 0$. □

Chebychev's inequality gives significantly better bounds compared to Markov's inequality. For example, it implies (compare with Equation 2.1), by substituting $a = \delta^2\mu^2$ in Equation 2.2.

$$\Pr[X \geq (1+\delta)\mu] \leq \Pr[(X-\mu)^2 \geq \delta^2\mu^2] \leq \frac{1}{\delta^2\mu} \tag{2.3}$$

The advantage of working with $(X-\mu)^2$ is that, we can use it to bound the probability of the "lower tail" also.

$$\Pr[X \geq (1-\delta)\mu] \leq \Pr[(X-\mu)^2 \geq \delta^2\mu^2] \leq \frac{1}{\delta^2\mu} \tag{2.4}$$

REMARK 2.1.8. Note that we did not use independence fully - we only needed that for all $i \neq j$, $\mathbb{E}[Y_iY_j] = \mathbb{E}[Y_i]\mathbb{E}_[Y_j]$. This is a strictly weaker condition than saying all random variables $Y_1, Y_2, \ldots Y_k$ are independent. Such random variables are called "pairwise independent". We will come across them later in the course.

We now present one of the stronger tools to estimate probability tails in the case of sum of fully independent random variables. In this case, the set up itself is about a random variable $X = \sum_{i=1}^{k} X_i$. Let $\mathbb{E}[X_i] = \mu_i$ and $\mathbb{E}[X] = \sum_i \mu_i = \mu$ (say). We have the following:

Chernoff Bound : For any $\alpha > \mu$, $\Pr[X \geq \alpha] \leq e^{\alpha-\mu}\left(\frac{\mu}{\alpha}\right)^{\alpha}$

In particular, it implies: $\Pr[X \geq (1+\delta)\mu] \leq \left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^{\mu}$

Proof of Chernoff Bound: Again the idea is to use Markov's inequality for an appropriately defined positive random variable. In this case, we will just use $Y = \left(\frac{\alpha}{\mu}\right)^X$ and it is related to the original probability that we wanted to estimate. More precisely,

$$\Pr[X \geq \alpha] = \Pr\left[\left(\frac{\alpha}{\mu}\right)^X \geq \left(\frac{\alpha}{\mu}\right)^{\alpha}\right] = \Pr\left[Y > \left(\frac{\alpha}{\mu}\right)^{\alpha}\right] \leq \frac{E[Y]}{\left(\frac{\alpha}{\mu}\right)^{\alpha}}$$

Since $\mathbb{E}[Y]$ is $\mathbb{E}\left[\left(\frac{\alpha}{\mu}\right)^X\right]$, we have the motivation to estimate $\mathbb{E}\left[a^X\right]$ in terms of $\mathbb{E}[X]$. Here we use the fact that $X = \sum_i [X_i]$ and are fully independent. Note that $\mathbb{E}\left[a^X\right] = \prod_i \mathbb{E}\left[a^{X_i}\right]$. Hence, we need to estimate $\mathbb{E}\left[a^{X_i}\right]$.

$$\begin{aligned}\mathbb{E}\left[a^{X_i}\right] &= a^0 \Pr[X_i = 0] + a^1 \Pr[X_i = 0] \\ &= 1.(1 - \mathbb{E}[X_i]) + a.\mathbb{E}[X_i] = a\mu_i + (1-\mu_i) = \mu_i(a-1) + 1 < e^{(a-1)\mu_i}\end{aligned}$$

In our context $a = \frac{\alpha}{\mu}$ and $Y = \left(\frac{\alpha}{\mu}\right)^X$. This gives,

$$\Pr[X \geq \alpha] \leq \frac{E[Y]}{\left(\frac{\alpha}{\mu}\right)^\alpha} \leq \frac{E\left[\left(\frac{\alpha}{\mu}\right)^X\right]}{\left(\frac{\alpha}{\mu}\right)^\alpha} \leq \frac{\prod_i E\left[\left(\frac{\alpha}{\mu}\right)^{X_i}\right]}{\left(\frac{\alpha}{\mu}\right)^\alpha} \leq \frac{\prod_i e^{(a-1)\mu_i}}{\left(\frac{\alpha}{\mu}\right)^\alpha} \leq \frac{e^{(\frac{\alpha}{\mu}-1)\mu}}{\left(\frac{\alpha}{\mu}\right)^\alpha} \leq e^{(\alpha-\mu)}\left(\frac{\mu}{\alpha}\right)^\alpha$$

□

2.1.3 Analysis of the Algorithm for MAXCUT

Recall Algorithm **??**. We want to guarantee that th expected size of the cut is at most half of the optimal cut size. In fact, we prove something stronger.

CLAIM 2.1.9. *Let X be the size of the cut output by the algorithm **??**. Then, $E[X] \geq \frac{m}{2}$ where m is the number of edges in the graph G.*

Proof. For each edge $e \in E$ define a random variable X_e as the following indicator variable:

$$X_e = \begin{cases} 1 & \text{if } e \in cut(S,T) \\ 0 & \text{otherwise} \end{cases}$$

By definition, $X = \sum_{e \in E} X_e$. Hence, by linearity of expectation:

$$E[X] = \sum_{e \in E} E[X_e] = mE[X_e]$$

We just need to notice that $E[X_e] = \Pr[X_e = 1] = \Pr[e \in cut(S,T)]$ because of tool 1. Notice that an edge e is in the cut if the two end points get into different sets among S and T. That is, out of the four possible outcomes of the random coin tosses correspnoding to the endpoint vertices of e, two of them leads to e being in $cut(S,T)$. Hence this is exactly $\frac{1}{2}$. This gives: $E[X] \geq \frac{m}{2}$. Hence the proof. □

Notice that the claim is stronger. Indeed, since the optimum cut can only cut at most m edges, the above also implies $E[X] \geq \frac{m}{2} \geq \frac{\text{OPTCUT}}{2}$.

2.2 Method of Conditional Expectation

We now describe the main technical idea to be learned this week. Mainly an algorithm specific technique of derandomization of randomized algorithms. This presentation is from Salil Vadhan's book on Pseudorandomness.

There are two kinds of randomized algorithms that we have seen so far - essentially to solve two kinds of problems. One is for the decision problems where the probability over different paths of the computation tree, the algorithm being correct is at least $\frac{2}{3}$. The other is for optimization problems where the expected size of the output has guarantees. The method of derandomization that we are going to discuss can in principle be applied for both, if the randomized algorithm in

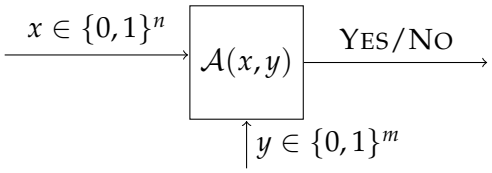
question uses the random bits in a peculiar way that certain measures can be computed efficiently about the output for particular settings of the randomness.

Main Idea: (as we described in the lecture) - we discuss the first kind of algorithms first and then adapt it to the second type. In the decision problem case, we know that $\frac{2}{3}$-rd paths in the computation tree are going to make the algorithm answer correctly. A vague idea would be - walk down the path of the tree, *making a choice deterministically and efficiently at each node (without trying both choices which leads to exponential time) maintaining the invariant that within the subtree that have restricted ourselves to, a $\frac{2}{3}$ fraction of paths within the subtree still make the algorithm go correct*. If we make choices like this and set the random bits based on that choices, it is intuitive that we will reach a leaf that makes the algroithm answer correctly and then we can just run the algorithm on that leaf (that choice of random bits) and output the answer. The process is deterministic and effcient and hence gives a derandomization of the original algorithm.

Of course, this is easier said than done. An important question remains - *at an intermediate node, in the above walk-down, how do we deterministically and efficiently decided whether to take the edge labelled* 0 *or* 1 *to move to the child*? Formalizing this will require us to fix the type of problem as decision vs search/optimization problem.

2.2.1 Framework for Algorithms for Decision Problems

Recall the following notational set up where $\mathcal{A}$ is the randomized algorithm solving a decision problem.



$$\forall x \in \{0,1\}^n, \Pr_{y \in \{0,1\}^m}[A(x,y) \text{ is correct.}] \geq \frac{2}{3}$$

Since we have to keep track of the fraction of paths for a particular partial setting of the random bits (while analysing the walk-down of the tree at an intermediate stage) - we define the following notation:

For every $i \in [n]$, bits $r_1, r_2, \ldots r_i \in \{0,1\}$, define:

$$p(r_1, r_2, \ldots, r_i) = \Pr_{y \in \{0,1\}^m} \{A(x,y) \text{ is correct } \mid (y_1 = r_1) \wedge (y_2 = r_2) \wedge \ldots \wedge (y_i = r_i)\}$$

Indeed, if we are at a particular node represented by a partial assignments $r_1, r_2, \ldots r_i$, the value of $p(r_1, r_2, \ldots, r_i)$ is the average of the value at the two children of that node since the bit is chosen uniformly at random. In terms of expectation, this is equivalent to :

$$p(r_1, r_2, \ldots r_i) = E_{y_{i+1} \in \{0,1\}}(r_1, r_2, \ldots r_i, y_{i+1})$$

To understand this definition clearly, let us ask, a first question, what is the value of $p(r_1, r_2, \ldots r_m)$ for a setting $r_1, r_2, \ldots r_m \in \{0,1\}$. (Class answered 0 or 1 depending on whether the setting represents a path which makes $\mathcal{A}$ correct or not). How about $p(\phi)$, which represents

the value of the function when no bit is set. Clearly, by definition, this represents the top of the computation tree, and hence the fraction of correct paths under the node is exactly the success probability of the algorithm. That is, $p(\phi) \geq \frac{2}{3}$.

Indeed, if we call $p(r_1, r_2, \ldots r_i) = \mu$, we have that: $E_{y_{i+1} \in \{0,1\}} p(r_1, r_2, \ldots r_i, y_{i+1}) = \mu$. Hence we know that there must exist a setting of y_{i+1} such that the value of the random variable - which in this case is $p(r_1, r_2, \ldots r_i, y_{i+1})$ - is at least the expected value. That is,

$$\exists r_{i+1} \in \{0,1\} : p(r_1, r_2, \ldots r_{i+1}) \geq p(r_1, r_2, \ldots r_i)$$

Applying this repeatedly, we have that $\exists r_1, r_2, \ldots r_m \in \{0,1\}$:

$$p(r_1, r_2, \ldots r_m) \geq p(r_1, r_2, \ldots r_{m-1}) \geq \ldots \geq p(r_1, r_2) \geq p(r_1) \geq p(\phi) \geq \frac{2}{3}$$

Notice that, by our observation, the left-end term is Boolean, and hence it must be that there exists $r_1, r_2, \ldots r_m \in \{0,1\}$ such that $p(r_1, r_2, \ldots r_m) = 1$. But then, this is not a big deal in the end, we knew about existance of such r_i's anyway. So in the end it does not look very useful.

But the above framework has an interesting feature. It also shows how to construct r_i's bit-by-bit, if we have an efficient algorithm to compute $p(r_1, r_2, \ldots r_i)$ for any i. Suppose we have computed $r_1, \ldots r_i$ already, we can compute r_{i+1} as follows: compute $p(r_1, r_2, \ldots, r_i, 0)$ and $p(r_1, r_2, \ldots, r_i, 1)$ using the above algorithm and set r_{i+1} to be whichever bit in $\{0,1\}$ which achieves the maximum.

However, we still have the problem of computing $p(r_1, r_2, \ldots r_i)$ for any $i \in [m]$ efficiently. This is where the algorithm-specifics come in. The algorithm $\mathcal{A}$ should be using the random bits in a peculiar way such that the value of $p(r_1, r_2, \ldots, r_i)$ can be computed efficiently for that algorithm $\mathcal{A}$. Indeed, the trivial method of computing $p(r_1, r_2, \ldots r_i)$ ends up taking exponential time in the worst case. Hence one has to use the algorithm-specific attributes to design this algorithm. We will demonstrate this in the next context.

2.2.2 Framework for Algorithms for Optimization Problems

We now the adapt the above framework for search and optimization problems. The guarantee for algorithms solving such problems is as follows - the expected size of the output is at least as "good" as this, where "good" means at most or at least in minimization and maximization problems respectively. For demonstrative purposes, we restrict ourselves to the randomized algorithm for MAXCUT that we presented earlier. Notice that the algorithm uses exactly n random bits where $|V| = n$. Note that S and T are the two subsets output by the algorithm. For any $i \in [n]$, define:

$$V(r_1, r_2, \ldots r_i) = \mathbb{E}_{y \in \{0,1\}^n} \left[|cut(S,T)| : (y_1 = r_1) \wedge (y_2 = r_2) \wedge \ldots \wedge (y_i = r_i) \right]$$

Similar to the previous setting, note that $V(\phi) \geq \frac{|E|}{2}$ since the expected size of the cut when no random bit is conditioned is similar to the original analysis of the algorithm. We apply the averaging principle and argue in a similar way that there must exist a choice of the random bits $r_1, r_2, \ldots, r_n \in \{0,1\}$ such that $cut(S,T)$ output by the algorithm has at least $\frac{|E|}{2}$ many edges.

Indeed, we start with $V(\phi) \geq \frac{|E|}{2}$. By averaging principle, there must exist $r_1 \in \{0,1\}$ such

that $V(r_1) \geq V(\phi) \geq \frac{|E|}{2}$. Continuing in a similar way there must exist $r_1, r_2, \ldots, r_n \in \{0,1\}$ such that :

$$V(r_1, r_2, \ldots r_{n-2}, r_{n-1}, r_n) \geq V(r_1, r_2, \ldots r_{n-2}, r_{n-1}) \geq V(r_1, r_2, \ldots r_{n-2}) \ldots \geq V(r_1) \geq V(\phi) \geq \frac{|E|}{2}$$

Again, this is not new information. We can always derive by a globally applying averaging principle that there must exist such a choice of random bits. But the advantage here is that if there is an efficient algorithm for computing $V(r_1, r_2, \ldots r_i)$ for any i, then we can find out the explicit choice of values of the bits as well. As in the previous case, if $r_1, r_2, \ldots r_i$ is already fixed, then we compute r_{i+1} as : compute $V(r_1, r_2, \ldots r_i, 0)$ and $V(r_1, r_2, \ldots r_i, 1)$ and set r_{i+1} to be that value in $\{0,1\}$ which results in the maximum among the two.

Computing $V(r_1, r_2, \ldots r_i)$ for the algorithm for MAXCUT : To apply the above framework, all we need is an efficient algorithm to compute the value of $V(r_1, r_2, \ldots r_i)$ for any choice of i and $r_1, r_2, \ldots r_i \in \{0,1\}$. Note that this is a speciality of the algorithm - more importantly the way the bits $y_1, y_2, \ldots y_n$ are used by the algorithm.

At any intermediate point of computation, there are vertices for which the decision (of whether they should be in the set S or not) is already made by then and there are vertices which are decided later. To keep track of this, we define:

$$\begin{aligned} S_i &= \{j \in [n] \mid j \leq i, b_j = 1\} \\ T_i &= \{j \in [n] \mid j \leq i, b_j = 0\} \\ U_i &= \{j \in [n] \mid j > i\} \end{aligned}$$

The algorithm will grow S_i to S and T_i to T, by randomly choosing the remaining vertices (U_i) to be in S or T. We need to compute the expected size of the cut conditioned on the fact that the sets S_i and T_i are already fixed by the algorithm. An immediate observation is that the edges that go across S_i and T_i will necessarily a part of the cut, since their endpoints are already at S and T respectively by definition. The edges which are fully within S_i or fully within T_i are not going to be a part of the cut finally. But there may be more number of edges which forms a part of the final cut. Considering this, we can write:

$$V(r_1, r_2, \ldots r_i) = |cut(S_i, T_i)| + \frac{|(cut(S_i, U_i)| + |cut(U_i, T_i)| + |cut(U_i, U_i)|}{2}$$

We need to explain the second term in the RHS. Consider edges $e = (u, v) \in E$ that has one endpoint $u \in S_i$ and the other endpoint in $v \in U_i$. Note that $u \in S$ finally, and hence (u, v) edge will be counted in the cut, if v falls into T. Since this is decided by choosing a random bit, the probability that the edge appears in the cut finally is $\frac{1}{2}$. Hence, the expected number of edges in $cut(S_i, U_i)$ which appear in the final cut is $\frac{|cut(S_i, U_i)|}{2}$. Similar argument explains the term $\frac{|cut(T_i, U_i)|}{2}$. To see the $\frac{|cut(U_i, U_i)|}{2}$ term, consider edges which are having both end points in U_i. They are both going to be put in S or T uniformly at random - hence out of the four possible outcomes (for these two vertices), two of them puts them in the final cut and two of them puts them outside the final cut output by the algorithm. Hence for any edge in $cut(U_i, U_i)$, with probability $\frac{1}{2}$ it will form a

part of the cut. That is, expected number of edges that gets contributed to the final cut is $\frac{|cut(U_i,U_i)|}{2}$. Hence the expression for $V(r_1, r_2, \ldots r_i)$ is correct.

Exercise 2.2.1 (See Problem Set 1(Problem 3))**.** In the derandomization of MAXCUT algorithm that we described, we derived an expression for $V(r_1, r_2, \ldots r_i)$ for any i and $r_1, r_2, \ldots r_i \in \{0, 1\}$. We used this to determine, the value of r_{i+1} by computing $V(r_1, r_2, \ldots r_i, 0)$ and $V(r_1, r_2, \ldots r_i, 1)$ and then choosing the value of r_{i+1} to be the one which produces the largest among the two. Prove that the choice of r_{i+1} will be 1 if vertex $i+1$ has more neighbors in T_i than in S_i and vice versa. Hence, write down the derandomized 0.5-approximation deterministic polynomial time algorithm for MAXCUT as a simple greedy algorithm in terms of the above rule.

Exercise 2.2.2 (See Problem Set 1(Problem 4))**.** Let $x_1, x_2, \ldots x_n \in \{0, 1\}$ be Boolean variables and let f be a Boolean formula in CNF form. That is, $f = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ where each C_i (called a *clause*) is a disjunction of literals in the set $\{x_1, \overline{x_1}, x_2, \overline{x_2}, \ldots, x_n, \overline{x_n}\}$. We want to find an assignment of the Boolean variables that satisfies as many clauses in the formula as possible.

(a) Write down a randomized algorithm that outputs an assignment with the guarantee that the expected number of clauses satisfied is at least $\frac{m}{2}$.

(b) Derandomize this algorithm using the method of conditional probabilities discussed in class to get a deterministic algorithm that satisfies at least $\frac{m}{2}$ number of clauses.

(c) Suppose k is the minimum number of literals in any clause, how will you modify the parameters in part(a) and (b)

2.3 Method of Pessimistic Estimators

We now see a more sophisticated adaptation of the method of conditional expectation. For this we need another randomized algorithm, as it is again going to be algorithm specific[7].

2.3.1 Congestion Minimization Problem

The problem that we are going to use as the example is called CONGESTION MINIMIZATION problem. We are given a directed graph G with k pairs of vertices $(s_1, t_1), (s_2, t_2), \ldots (s_k, t_k)$ which are sources and destinations respectively. We want to route packets from sources to destination through edge disjoint paths in the graphs so that there is no edge that is used for two paths and hence no change of a congestion. However, even testing whether there are edge-disjoint paths between such pairs of vertices is NP-hard for directed graphs even for $k = 2$. Hence, we have to allow congestion, but ideally we would like to minimise congestion. Formally, a collection of paths $P_1, P_2, \ldots P_k$ (which need not be vertex disjoint) in the above problem is a solution with congestion C, if every edge takes part in at most C paths in the collection. Indeed, we would like to find out a collection of paths with least congestion. The case when $C = 1$ is exactly the edge disjoint path problem which indicates that the optimization problem is hard to solve.

Next step is to look for approximation algorithms. We would like to design algorithm which outputs a set of paths with a guarantess that the congestion is at most αC where C is the optimal

[7] That is, it can be applied for a class of algorithms for the problem which has the property.

congestion possible for the given graph and the $\{(s_i, t_i)\}_{1 \leq i \leq t}$ pairs. Indeed, the problem admits a randomized approximation algorithm with a reasonable approximation ratio.

THEOREM 2.3.1. CONGESTION MINIMIZATION *admits a randomized algorithm where the solution is guaranteed to be at most* $\left(\frac{\log n}{\log \log n}\right) \times OPT$ *where OPT is the optimal congestion possible for the input instance and n is the size of the graph.*

We need some details about this randomized algorithm and the analysis since we have to deal with specifics of the analysis in the method itself.

2.3.2 Randomized Approximation Algorithm

The algorithm uses a standard and quite effective technique of designing randomized algorithms which is *linear programming relaxation and randomized rounding*. Firsly the problem can be written as a linear program as follows:

LP Formulation: For any $i \in [k]$, let $\mathcal{P}_i$ denote the set of paths in the graph G which are from vertices s_i to t_i. In the solution, exactly one of these paths needs to be chosen. Let P be the notation for one such path. Let us introduce a Boolean variable x_i^P to indicate whether the path $P \in \mathcal{P}_i$ is chosen for the solution or not. The following k constraints say, that from each $\mathcal{P}_i$ *exactly one* path must be chosen for the solution :

$$\forall i \in [k], \sum_{P \in \mathcal{P}_i} x_I^P = 1 \tag{2.5}$$

And, we need to minimise the congestion bound C subject to above constraints. Writing down this objective function mathematically:

$$C = \sum_{e \in E} \sum_{\substack{P \in \mathcal{P}_i \\ e \in P}} x_i^P \tag{2.6}$$

Unfortunately, solving such "integer linear programs" is NP-hard. But a technique is that of linearprogramming relaxation, where we do not insist anymore that the variables must take Boolean values. But instead, we allow them to be real numbers under the constraint - $\forall i, P, 0 \leq x_i^P \leq 1$ which are again linear constraints. There are standard techniques by which this can be solved now in polynomial in the number of variables. However, note that the number of variables in our setting is exponential in n. But we will ignore this fact[8] for now as the exposition of the technique is easier with the above set up.

Randomized Rounding: The solution to the above linear program gives us values for x_i^P (call them α_i^P between 0 and 1) for $i \in [k]$ and $P \in \mathcal{P}_i$ such that the congestion expression is at most C^* (and is optimal). But this does not point to choice of any path $P \in P_i$ as they are not Booelan variables. So we need to make that choice (and hence turn th e values to Boolean) and this is the

[8]The trick to address this issue is to formulate a linear program with the granularity of edges - that is introducing variables at the edge-level than path-level as we have done.

place where randomness is used[9].

The idea is as follows. For each i, we will choose a path $P \in \mathcal{P}_i$ at random with probability x_i^P. Notice that this is a probability distribution because of contstraint 2.5. This can also be seen as, for each i, we will choose one of the x_i^P at random with probability α_i^P and then assign that variable to 1 and make the rest of the variables for that i to be 0. This is equivalent to choosing one path $P_i \in \mathcal{P}_i$ to be in the solution. For example, say we have three paths with values $0.2, 0.7$, and 0.1. Get a random number between 0 and 1. If the number is between 0 and 0.2, pick the first path. If the number is between 0.2 and 0.9, pick the second path, and if the number is between 0.9 and 1, pick the third path.

Bounding the Approximation: We need to analyse how badly the objective function (which was evaluating to optimal value C when α_i^P were the assignments for x_i^Ps) be affected by this *rounding* step. For every edge $e \in E$, define a random variable Y_i^e as follows:

$$Y_i^e = \begin{cases} 1 & \text{if } e \in P_i \\ 0 & \text{otherwise} \end{cases} \qquad \text{and by definition, } \mathbb{E}[Y_i^e] \leq \Pr[e \in P_i] \leq \sum_{\substack{P \in \mathcal{P}_i \\ e \in P}} \alpha_i^P$$

Define $Y_e = \sum_i Y_i^e$. By linearity of expectation, $\mathbb{E}[Y_e] = \sum_i \mathbb{E}[Y_i^e] = C^*$ (because it is exactly the objective function). Now we are in a perfect situation for a tail bound. What is the probability that the random variable (which in this case is a sum of independent random variables).

PROPOSITION 2.3.2 (Chernoff Bound). *Let $X = \sum X_i$ be a random variable expressed as a sum of X_i's which are independent random variables. Let $\mathbb{E}[X] \leq \mu$. Then for any $\alpha > 1$,*

$$\Pr\left[X \geq \alpha\mu\right] \leq e^{-\mu \times [\alpha \ln \alpha - \alpha + 1]}$$

We just need to apply this with a $\alpha = \frac{\log n}{\log \log n}$ is the approximation ratio that we are looking for. Working out the math, this gives $\Pr[Y_e > \alpha C^*] \leq \frac{1}{n^3}$. By union bound, probability that there exists an edge $e \in E$ with $Y_e > \alpha C^*$ is at most $\sum_e \Pr[Y_e > \alpha C^*] \leq \frac{1}{n}$. In other words, with probability at least $1 - \frac{1}{n}$ the above rounding gives a solution (equivalently a Boolean assignment for the variables) which satisfies all the constraints but at the same time does not make the objective function value worse that a factor of α. Hence it is a randomized alpha approximation algorithm.

2.3.3 Pessimistic Estimator

We abstract out a scenario from the above analysis as follows. Let $X = X_1 + X_2 + \ldots + X_k$ be a random variable that is expressed as a sum of independent random variables in $[0, 1]$. Chernoff Bound is typically applied when we want to estimate $\Pr[\sum_i X_i > \alpha]$. To see the event as a Boolean

[9] A simple deterministic idea is worth thinking about - namely, for every i, choose the P for which α_i^P is maximum to be the path - this amounts to assigning x_i^P to one for that path P but 0 for the other paths. But this can increase the congestion (constraint **??**) without any reasonable bound.

value, define the indicator function:

$$I(x_1, x_2, \ldots x_n) = \begin{cases} 1 & \text{if } \sum_i x_i \geq \alpha \\ 0 & \text{otherwise} \end{cases}$$

So one way to descend down the tree to derandomize is to apply the method of conditional expectation on $I(x_1, x_2, \ldots x_n)$. Similar to evaluating $e(r_1, r_2, \ldots r_i)$ in the MAXCUT algorithm, we should be able to efficiently evaluate:

$$\mathbb{E}[I(x_1, x_2, \ldots, x_i, X_{i+1}, \ldots X_n)] = \Pr\left[\sum_{j=1}^{k} X_j \geq \alpha \mid X_1 = x_1, X_2 = x_2, \ldots X_i = x_i\right]$$

But unfortunately, the RHS expression is not easy to compute since it depends on the distribution of the X_j variables. However, we can still work with the framework, with the idea of replacing it with an upper bound function instead. Since we are bounding a bad event of the sum being large, a function which gives a larger value would be more "pessimistic" and hence the term *pessimistic estimator*.

We can use an exponentiation idea coming from Chernoff Bound proof: For any $t > 0$:

$$\Pr\left[\sum_{j=1}^{k} X_j \geq \alpha\right] \leq \mathbb{E}\left[\frac{\prod_{j=1}^{k} e^{tX_j}}{e^{t\alpha}}\right]$$

This upper bound is easy to compute when you have substitutions for some of the X_is. That is,

$$\Pr\left[\sum_{j=1}^{k} X_i \geq \alpha \mid X_1 = x_1, X_2 = x_2, \ldots X_i = x_i\right] \leq \left(\frac{\prod_{j=1}^{i} e^{tx_i}}{e^{t\alpha}}\right) \mathbb{E}\left[\prod_{j=i+1}^{k} e^{tX_i}\right]$$

Since we know the distribution of each X_j, we can compute the $\mathbb{E}\left[\prod_{j=i+1}^{k} e^{tX_i}\right]$ in the right hand side as we do in the proof of Chernoff Bound proof itself.

Applying the Method to the Algorithm for Congestion Minimization: Recall the random variables involved. For every edge $e \in E$, we used random variable $Y_i^e = \begin{cases} 1 & \text{if } e \in P_i \\ 0 & \text{otherwise} \end{cases}$ Define $Y_e = \sum_i Y_i^e$. By linearity of expectation, $\mathbb{E}[Y_e] = \sum_i \mathbb{E}[Y_i^e] = C^*$. In the analysis of the algorithm, we proved, by using Chernoff bound that, for $\alpha = \frac{\log n}{\log \log n}$:

$$\Pr\left[\sum_i Y_i^e > \alpha C^*\right] \leq \frac{1}{n}$$

As per our plan, rather than working with the indicator $I(x_1, x_2, \ldots x_i)$ we will work with the pessimistic esimator, for edge $e \in E$: $\left(\frac{\prod_{j=1}^{i} e^{tr_i}}{e^{t\alpha}}\right) \mathbb{E}\left[\prod_{j=i+1}^{k} e^{tY_i^e}\right]$. We will also incorportate union bound into our estimate to use the pessiminstic estimator for the whole event as :

$$I(r_1, r_2, \ldots r_i) \leq f(r_1, r_2, \ldots r_i) = \left(\frac{\prod_{j=1}^{i} e^{tr_i}}{e^{t\alpha}} \right) \sum_{e \in E} \left(\mathbb{E} \left[\prod_{j=i+1}^{k} e^{tY_i^e} \right] \right)$$

By notation, we know that $f(r_1, r_2, \ldots r_n) \leq \frac{1}{n}$.

1: Jayalal says: Todo - A few more lines to be completed here about the precise application of the estimators.

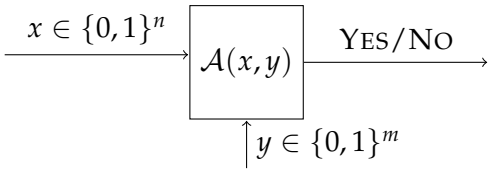
Amplification : Making Algorithms Err Less

Lecturer : Jayalal Sarma

WEEK 3

In the last week, we saw algorithm specific techniques of derandomization. More precisely, the derandomization uses the peculiarity of the algorithms in the way they use the random bits and the analysis. In this week, we will go back to the abstract set up where we know nothing about the algorithm other than the resource bounds it uses.

We recall some notations first. A randomized algorithm $\mathcal{A}$ on input x runs in time $t(n)$ (where $n = |x|$) and let $y \in \{0,1\}^{m(n)}$ be the concatenation of the unbiased coin toss experiement that the algorithm does during its execution. Notice that $m(n) \leq t(n)$ (we drop the n when it is not required explicitly). If the algorithm runs in polynomial time $t(n) \leq n^c$ for a constant c independent of n.



The guarantee we have is there is an $\epsilon \in (0, \frac{1}{2}]$.

$$\forall x \in \{0,1\}^n, \Pr_{y \in \{0,1\}^m}[A(x,y) \text{ is correct.}] \geq \frac{1}{2} + \epsilon$$

Imagine that we had a success probability of very close to 1. That is, $\epsilon > \frac{1}{2} - \frac{1}{2^m}$. That is, $\forall x \in \{0,1\}^n : \Pr_{y \in \{0,1\}^m}[A(x,y) \text{ is correct.}] > 1 - \frac{1}{2^m}$. Notice that, now the algorithm does not need to use randomness and can fix y to be any string in $\{0,1\}^m$ and run the algorithm $\mathcal{A}$ and the answer is guaranteed to be correct. (This is because, if there exists at least one $y \in \{0,1\}^m$ for which the algorithm errs then the success probability would have been $\leq 1 - \frac{1}{2^m}$.) Thus we would have derandomized the algortithm efficiently.

But how do we achieve such high success probability? Viewing a randomized algorithm as an experiment that we do in physics lab to compute a value, we will repeat the experiment and take the most frequent value in order to reduce the error. However, this also increases the number of random bits which goes against the above plan. Let us formally review this amplifcation method nevertheless.

3.1 Success Probability Amplifcation by Repetition

We first write down the algorithm which follows the simple idea of repetition with independent random bits.

Why would this improve the success probability? and if so, how does it depend on k?. The following lemma answers these. Fix the input x. Let $\mathcal{E}$ represent the event that $\mathcal{A}$ accept on the random string y.

Algorithm 3.6 ($\mathcal{A}'$) : input $x \in \{0,1\}^n$

1: $count \leftarrow 0$.
2: Choose k independent random strings $y_1, y_2, \ldots y_k \in \{0,1\}^m$. ▷ Uses $O(kn)$ random bits.
3: **for** `each` $i \in [k]$ **do**
4: If $\mathcal{A}(x, y_i)$ accepts, if so increment *count*
5: **end for**
6: If $[count > \frac{k}{2}]$ then output YES else output NO.

LEMMA 3.1.1. *If $\mathcal{E}$ is an event that $Pr(\mathcal{E}) \geq \frac{1}{2} + \epsilon$, then the probability the $\mathcal{E}$ occurs atleast $\frac{k}{2}$ times on k independent trials is at least $1 - \frac{1}{2}(1 - 4\epsilon^2)^{\frac{k}{2}}$*

Proof. Let q denote the probability the $\mathcal{E}$ occurs atleast $\frac{k}{2}$ times on k independent trials. Let q_i = Pr($\mathcal{E}$ occurs exatly i times in k trials), $0 \leq i \leq k$. Thus, $q = 1 - \sum_{i=0}^{\lfloor \frac{k}{2} \rfloor} q_i$. We will analyse the complementary event: Pr($\mathcal{E}$ occurs atmost $\frac{k}{2}$ times) = $\sum_{i=0}^{\lfloor \frac{k}{2} \rfloor} q_i$.
We show an upper bound on each q_i and thus show an lower bound on q.

$$\begin{aligned}
q_i &= \binom{k}{i}(\frac{1}{2}+\epsilon)^i(\frac{1}{2}-\epsilon)^{k-i} \\
&\leq \binom{k}{i}\left(\frac{1}{2}+\epsilon\right)^i\left(\frac{1}{2}-\epsilon\right)^{k-i}\left(\frac{\frac{1}{2}+\epsilon}{\frac{1}{2}-\epsilon}\right)^{\frac{k}{2}-i} \text{(because } \epsilon \leq \frac{1}{2}) \\
&= \binom{k}{i}\left(\frac{1}{2}+\epsilon\right)^{\frac{k}{2}}\left(\frac{1}{2}-\epsilon\right)^{\frac{k}{2}} \\
&= \binom{k}{i}\left(\frac{1}{4}-\epsilon^2\right)^{\frac{k}{2}}
\end{aligned}$$

Now we analyse the sum:

$$\begin{aligned}
\sum_{i=0}^{\lfloor \frac{k}{2} \rfloor} q_i &\leq \sum_{i=0}^{\lfloor \frac{k}{2} \rfloor}\binom{k}{i}\left(\frac{1}{4}-\epsilon^2\right)^{\frac{k}{2}} \\
q = 1 - \sum_{i=0}^{\lfloor \frac{k}{2} \rfloor} q_i &\geq \sum_{i=0}^{\lfloor \frac{k}{2} \rfloor}\binom{k}{i}\left(\frac{1}{4}-\epsilon^2\right)^{\frac{k}{2}} \\
&= 1 - \left(\frac{1}{4}-\epsilon^2\right)^{\frac{k}{2}} 2^{k-1} \\
&= 1 - \frac{1}{2}\left(1-4\epsilon^2\right)^{\frac{k}{2}} \\
\text{Thus, } q &\geq 1 - \frac{1}{2}\left(1-4\epsilon^2\right)^{\frac{k}{2}}
\end{aligned}$$

□

Thus, if we had an algorithm $\mathcal{A}$ with $\epsilon = \frac{1}{3}$ (that is success probability is at least $\frac{2}{3}$, and we want an algorithm with $\mathcal{A}'$ with $\epsilon = \frac{1}{4}$ (that is, success probability is at least $\frac{3}{4}$). Then, the number of

times the iteration that needs to be done can be back calculated as the k that satisfies:

$$1 - \frac{1}{2}\left(1 - \frac{4}{9}\right)^{\frac{k}{2}} \geq \frac{3}{4}$$

which will be a constant. Quite interestingly, we can do this even when the required error probability is exponentially small. That is, suppose we require the success probability to be $1 - \frac{1}{2^{q(n)}}$ which is quite close to 1. Then the value of k should be :

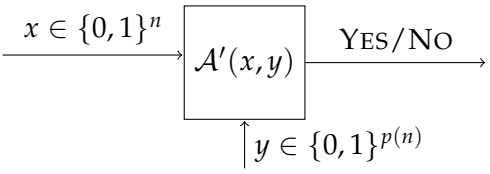
$$1 - \frac{1}{2}\left(1 - \frac{4}{9}\right)^{\frac{k}{2}} \geq 1 - \frac{1}{2^{q(n)}} \implies \left(\frac{9}{5}\right)^{\frac{k}{2}} \leq 2^{q(n)-1}$$

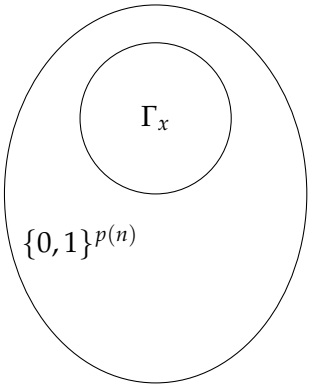
which in turn would imply $k = p(n)$ to be a polynomial value in terms of n. Thus we have the following lemma:

LEMMA 3.1.2 (**Amplification Lemma**). *Let $\mathcal{A}$ be a randomized algorithm running in time $\mathsf{poly}(n)$ which has $\frac{1}{2} + \epsilon$ as the probability of success. Then for any $q(n)$, we have a randomized algorithm $\mathcal{A}'$ that runs in time $\mathsf{poly}(n)$ with the following success probability. For every input $x \in \{0,1\}^n$.*

$$\Pr_y \left[\mathcal{A}'(x,y) \text{ is correct }\right] \geq 1 - 2^{-q(n)}$$

Define $\Gamma_x = \{y \in \{0,1\}^{p(n)} \mid \mathcal{A}(x,y) \text{ is correct }\}$.





The guarantee we have is : $\forall x,\ |\Gamma_x| \geq (1 - 2^{-q(n)})2^{p(n)}$

Exercise 3.1.3 (See Problem Set 1(Problem 5))**.** A decision problem L is in a class called BPP if there exists a randomized polynomial-time algorithm A such that for every $x \in L$ it holds that $\Pr[A(x,y) = 1] \geq \frac{2}{3}$, and for every $x \notin L$ it holds that $\Pr[A(x) = 0] \geq \frac{2}{3}$. For $\epsilon : \mathbb{N} \to [0,1]$, let BPP_ϵ denote the class of decision problems that can be solved in probabilistic polynomial time with error probability upper-bounded by ϵ. Prove the following two claims:

(a) For every positive polynomial p and $\epsilon(n) = \frac{1}{2} - \frac{1}{p(n)}$, the class BPP_ϵ equals BPP.

(b) For every positive polynomial p and $\epsilon(n) = 2^{-p(n)}$, the class BPP_ϵ equals BPP.

We already proved something similar in class (See Amplication Lemma). This exercise asks you to prove the same using tail bounds. Given an algorithm A, consider an algorithm A' that on input x invokes A on x for $t(|x|)$ times, and decided based on majority as we did in class. For Part (a) $t(n) = O(p(n)^2)$ and apply Chebyshev's Inequality. For Part 2 set $t(n) = O(p(n))$ and apply the Chernoff Bound.

3.2 Sipser's Argument

From the previous section, we concluded that, for every $x \in \{0,1\}^{p(n)}$, there is a large set Γ_x of random strings which are "good" for x. However, we do not know how to find even an element $y \in \Gamma_x$ in time $\mathsf{poly}(n)$. If we do, then we have a derandomization for the algorithm $\mathcal{A}'$ (which derandomizes $\mathcal{A}$ too).

It is intuitive to think that since a large fraction of $y \in \{0,1\}^{p(n)}$ are good for each $x \in \{0,1\}^n$, could there be a single y which is good for all x? We show that this is indeed the case. This, in fact, is a simple consequence of the amplification lemma in the previous section.

We first apply Lemma 3.1.2 with $q(n) = 2n$. Thus we have a randomized polynomial time algorithm with error bound 2^{-2n} (i.e. at most 2^{-2n} fraction of random strings are "bad") using $p(n)$ randomness. More precisely, we have that:

$$\text{Answer is YES for input } x \Rightarrow \Pr_y [\mathcal{A}(x,y) \text{ accepts }] \geq 1 - 2^{-2n} \tag{3.7}$$

$$\text{Answer is NO for input } x \Rightarrow \Pr_y [\mathcal{A}(x,y) \text{ accepts }] \geq 2^{-2n} \tag{3.8}$$

That is in such a machine the number of random strings y which lead the machine to output a wrong answer is bounded by 2^{-2n}. Now let us consider a matrix M whose rows are indexed by inputs of length n and columns are indexed by random strings of length $p(n)$, and the (x,y)th entry is 1 if on fixing the random bits to be y the machine M on input x outputs correctly and it is 0 otherwise. That is $M[x,y] = 1$ if and only if $\mathcal{A}'(x,y)$ is correct for the input x. By the amplification we are guaranteed that for a given input x at most 2^{-2n} fraction of the random strings can have $M[x,y] = 0$. Hence the total number of zeros in the A matrix is at most the number of rows times the maximum number of zeros in a row, which is equal to

$$\begin{aligned} \#\,0\text{'s in matrix } M &\leq 2^n \times 2^{-2n} \times 2^{p(n)} \\ &\leq 2^{p(n)-n} \end{aligned}$$

But the total number of zeros, $2^{p(n)-n}$ is strictly less than the number of columns in the matrix M. Hence there must be at least one column with no zeros in it. If a column in the M matrix has no zeros then by the definition of M matrix, the random string w represented by this column when fed as random bits to machine $\mathcal{A}'(x,w)$ would output the correct answer for every $x \in \{0,1\}^n$. Thus we have the follwing lemma.

LEMMA 3.2.1. *For every n, there is a $y \in \{0,1\}^{p(n)}$ such that $y \in \Gamma_x$ for every $x \in \{0,1\}^n$.*

Thus there is a function $h : \mathbb{N} \to \Sigma^*$ such that the answer to x is YES if we run the algorithm $\mathcal{A}$ on input x and random string $h(|x|)$ it is guaranteed not to err. This will give a complete deranomization. However, we need the function h to be computable in polynomial time and that is a challege.

3.3 Amplification with Dependent Trials using Expanders

We study another seemingly unrelated graph theoretic object which are called expanders. They have been extensively studied and used in many areas of science and engineering as the set of sparse graph families which has high connectivity connectivity properties. These are particularly desirable in network design such that the network is highly fault tolerant - even the failure of a few links (edges) will not affect the connectivity of the network. The fact that this can be achieved without having too many edges (complete graph is a trivial way to do this) is what makes expander graphs more applicable in this setting.

Expanders have been studied in the theoretical side as well for past few decades and have found numerous applications. Informally, the graph is such that every subset of vertices fetches a large set of vertices in its immediate neighborhood. For a set of vertices in a graph G, define $N(S)$ to be the neighbors of V. We define the graph class formally now.

DEFINITION 3.3.1. *(**Expander Graphs)** A graph $G(V, E)$ is said to be a (α, β)-expander if every $S \subseteq V$ such that $|S| \leq \alpha|V|$, the number of new neighbors $|N(S) \setminus S| \geq \beta|S|$.*

A trivial (but unfortunately useless) example of an expander graph is the complete graph. Indeed, in a complete graph on n vertices, if we choose any $S \subseteq V$, such that $|S| \leq \frac{n}{2}$, we have that $N(S) \setminus S = V \setminus S$. Hence, $|N(S) \setminus S| \geq |S|$. This gives that it is a $(\frac{1}{2}, 1)$-expander as per the above definition. However, it is going to be useless as we will see in our application. In fact, ideally, we would like expander graphs where the degree d and α, β are all constants independent of n.

Thus, we are looking for graphs with constant degree (if possible, even regular). It implies that the graph must be sparse[10] Let us also record a non-example of an expander, which is even a sparse graph. Consider an $n \times n$ complete grid graph, which has n^2 vertices and all possible *grid edges* present. Consider a set S which is of size $\sqrt{n}$, which has n vertices and hence $|S| \leq \alpha|V|$ for any constant α. However, the number of new neighbors for the set in the grid graph will be $O(\sqrt{n})$ which is not at most $\beta|S|$ for any constant β. Hence such a graph is not an expander for any constant α and β.

To ensure that the definition of expander is practiced enough, we suggest the following exericse which derives a combinatorial consequence of the expansion property.

Exercise 3.3.2 (See Problem Set 1(Problem 6))**.** Let $G(V, E)$ be an undirected graph n vertices which is $(\frac{1}{2}, 2)$-expander. Show that the diameter of the graph is at most $O(\log n)$. The diameter of a graph is at most k if and only if between any two vertices in the graph G, there is a path of length at most k in the graph G.

Random Walks on Expanders and Our Application: Although the above definition explains the name expander graphs, the application in our context comes from the fact that if we choose a vertex in the expander at random and then choose the next vertices uniformly at random, the distribution of the ℓ^{th} vertex that you get to (as ℓ increases) is very close to uniform over the entire connected component of the graph where your starting vertex was lying. Intuitively, this is termed as the *rapid mixing* of random walks on expander walks.

[10] a graph is said to be sparse if $|E| = O(|V|)$, or else it is called dense.

We will prove the technical details of this idea at a later point in the course where we introduce expander graphs. Informally, to apply the expander graphs in the context of the success probability amplification, we consider the graph on G as a graph on $2^{p(n)}$ vertices indexed by $\{0,1\}^{p(n)}$. That is, each vertex in the graph can be interpreted as a $y \in \{0,1\}^{p(n)}$. Based on this, the following algorithm gives a good amplification of success probability.

Algorithm 3.7 ($\mathcal{A}'$) : input $x \in \{0,1\}^n$

1: $count \leftarrow 0$.
2: Let $G(V,E)$ be an expander graph on $2^{p(n)}$ vertices.
3: Choose a vertex $y_1 \in V$ uniformly at random. ▷ Uses $O(p(n))$ random bits
4: Starting at v perform a random walk for k steps in G. Let $y_1, y_2, \ldots y_k \in \{0,1\}^m$ be the vertices representing the walk. ▷ Uses $O(k \log d)$ random bits.
5: **for** `each` $i \in [k]$ **do**
6: If $\mathcal{A}(x, y_i)$ accepts, if so increment *count*
7: **end for**
8: If $[count > \frac{k}{2}]$ then output YES else output NO.

The idea of the above algorithm is based on the rapid mixing of random walks. Without the details, the strings $y_1, y_2, \ldots y_k$ are "almost" as good as randomly chosen y_i's since the random walk mixes fast and hence the amplification (although with weaker parameters) can be derived. This will be done in upcoming lectures.

Notice that the number of random bits used by the algorithm is $O(k \log d)$. This explains why the complete graph (whose degree in this case would have been $2^{p(n)} - 1$) would not have been good enough for our purpose because it leads to an exponential running time for our algorithm.

3.4 Amplification for One-sided Error Algorithms using Dispersers

Now we handle the case of one-sided error algorithms. These algorithms have the following peculiarity. For a one-sided error algorithm $\mathcal{A}$:

$$\text{Answer is YES for input } x \Rightarrow \Pr_y [\mathcal{A}(x,y) \text{ accepts }] \geq \frac{1}{2} + \epsilon \tag{3.9}$$

$$\text{Answer is NO for input } x \Rightarrow \Pr_y [\mathcal{A}(x,y) \text{ accepts }] = 0 \tag{3.10}$$

The success probability amplification of such algorithms is easier.

Algorithm 3.8 ($\mathcal{A}'$) : input $x \in \{0,1\}^n$

1: Choose k independent random strings $y_1, y_2, \ldots y_k \in \{0,1\}^m$. ▷ Uses $O(km)$ random bits.
2: **for** `each` $i \in [k]$ **do**
3: If $\mathcal{A}(x, y_i)$ accepts, ACCEPT.
4: **end for**
5: REJECT.

LEMMA 3.4.1. *If $\mathcal{A}$ has success probability at least $1 - \epsilon$ then $\mathcal{A}'$ has success probability at least $1 - \epsilon^k$.*

Proof. The argument is rather straightforward. Suppose the answer is NO, then we know that no matter what y_i we choose, $\mathcal{A}(x, y_i)$ rejects and hence the algorithm $\mathcal{A}'$ rejects. Suppose the answer is YES, then $\mathcal{A}'$ rejects (which is an error), if none of the y_i chosen makes $\mathcal{A}(x, y_i)$ accept in step 3. Since each y_i is chosen uniformly at random:

$$Pr[\forall i, \mathcal{A}(x, y_i) \text{ rejects }] = \prod_i (Pr[\mathcal{A}(x, y_i) \text{ rejects }]) \leq \epsilon^k$$

Hence, the probability that $\mathcal{A}'$ accepts is at least $1 - \epsilon^k$. □

Again, we can use the ideas from the previous section on expanders in order to do more randomness efficient amplification of success probability. However, since the algorithm is one-sides, we have other methods too. We now define a new combinatorial object called *disperser* which is useful for amplifying the success probability of an algorithm which has one-sided error, using much less number of random bits.

DEFINITION 3.4.2 (**Dispersers**). *A bipartite graph $G = (U \cup V, E)$ is called an $(1 - \epsilon)$-disperser with threshold T if, for all $S \subseteq U$ of size $|S| \geq T$, we have that the size of the set of neighbors of S in V,$|\Gamma(S)|$, is more than $\epsilon|V|$.*

Notice the difference between the definition of expanders and dispersers. Here a large set should have large fraction on the right side as the neighbors. Intuitively, disperser will "disperse" from a large to set to a good fraction of vertices on the right side.

We will now use a disperser to do randomness efficient amplification for one sided error algorithms. Notice that the algorithm $\mathcal{A}$ is using r random bits.

THEOREM 3.4.3. *Suppose we have an explicit $\frac{1}{2}$-disperser with threshold T, degree d, $U = \{0,1\}^R$ and $V = \{0,1\}^m$. Then a one-sided error algorithm running in time t, using r random bits to achieve error probability $\frac{1}{2}$ can be converted to a one-sided error algorithm, running in time $(R + d + t)^{O(1)}$ using R random bits to achieve error probability $\frac{T}{2^R}$.*

Proof. We write down a formal proof of the above theorem. Suppose the algorithm that we start with is $\mathcal{A}$ with success probability $\frac{1}{2}$. The amplification algorithm is as follows. Suppose we have an explicit disperser with the above parameters. For a vertex $y \in U$, denote $N(y)$ to be the neighbors of y. Indeed, $N(y) \subseteq V$.

Algorithm 3.9 ($\mathcal{A}'$) : input $x \in \{0,1\}^n$

1: Choose $y \in U = \{0,1\}^R$ uniformly at random. ▷ Uses R random bits.
2: Let $N(y)$ be $\{y_1, y_2, \ldots y_d\}$.
3: **for** `each` $i \in [d]$ **do**
4: If $\mathcal{A}(x, y_i)$ accepts, ACCEPT.
5: **end for**
6: REJECT.

We analyse the probability of error in the YES case. For the algorithm $\mathcal{A}$ consider the following set

of good random strings.

$$\Gamma_x = \{y \in \{0,1\}^m \mid \mathcal{A}(x,y) \text{ says YES }\}$$

From the fact that the success probability of $\mathcal{A}$ is at least $\frac{1}{2}$, we have $|\Gamma_x| \geq \frac{1}{2}|U|$. Notice that for any $y \in \{0,1\}^R$, y is bad for $\mathcal{A}'$ if none of $N(y) = \{y_1, y_2, \ldots y_d\}$ is in Γ_x. Define $S \subseteq U$ as follows.

$$S = \{y \in \{0,1\}^R \mid N(y) \cap \Gamma_x = \phi\}$$

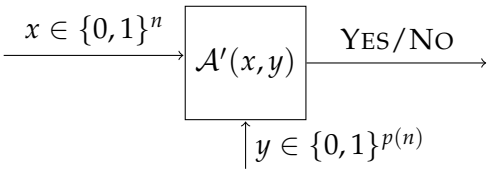
Suppose the error probability is more than $\frac{T}{2^R}$. This implies that $|S| > T$. However, $N(S) \cap \Gamma_x \neq \phi$. This set S contradicts the definition of disperser. Hence the proof. □

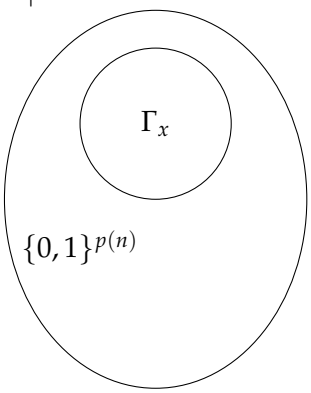
3.5 Deranomization of One-sided Error Algorithms using Hitting Set Generators

We now give a method of completely derandomizing One-sided error algorithms using the combinatorial objects called hitting sets.

DEFINITION 3.5.1 (**Hitting Sets**). *Let $\mathcal{C}$ be a collection of subsets over a finite universe U. A $(1-\epsilon)$-hitting set for $\mathcal{C}$ is a set $H \subseteq U$ such that for every $S \in \mathcal{C}$ of size at least $(1-\epsilon)|U|$, we have: $S \cap H \neq \phi$*

What is the connection to deranomization. Define $\Gamma_x = \{y \in \{0,1\}^{p(n)} \mid \mathcal{A}(x,y) \text{ is correct }\}$. If the success probability of $\mathcal{A}$ is at least $1-\epsilon$, we have $|\Gamma_x| \geq (1-\epsilon)|U|$.





Hence a hitting set in the above definition is usefu l, if the collection of sets is the set of possible Γ_x for every x of length n and for every algorithm $\mathcal{A}$ which runs in polynomial time.

This motivates the following definition.

DEFINITION 3.5.2 (**Hitting Set Generator**). *Let $\mathcal{C}$ be a collection of subsets over a finite universe U. For $\mathcal{C}$, a $(1-\epsilon)$-hitting set generator is a deterministic algorithm which on input r, the set $H_r \subseteq \{0,1\}^r$ which is a $(1-\epsilon)$-hitting set for $\mathcal{C}$.*

If we consider $\mathcal{C}$ to be the set of possible $\Gamma_x \subseteq \{0,1\}^r$ for every x of length n and for every algorithm $\mathcal{A}$ which runs in polynomial time. From the definitions itself the following theorem follows.

THEOREM 3.5.3. *If there is a $\frac{1}{2}$-hitting set generator that runs in time $\mathsf{poly}(n)$ for the above set system, then any one-sided error randomized algorithm that has success probability at least $\frac{1}{2}$ has an equivalent deterministic polynomial time algorithm.*

Proof. The deterministic algorithm is as follows:

Algorithm 3.10 ($\mathcal{A}'$) : input $x \in \{0,1\}^n$

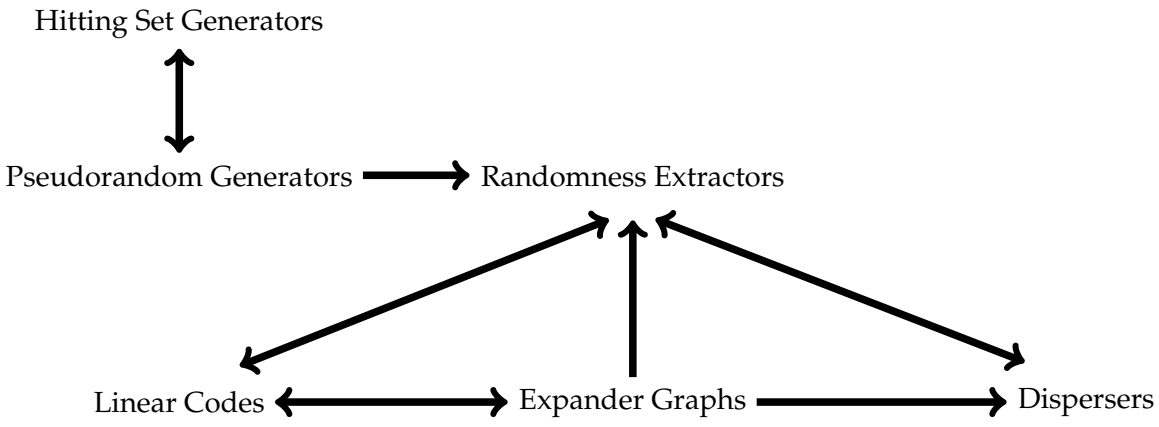
1: Using the generator, compute $H_r = \{y_1, y_2, \ldots y_\ell\}$ ▷ Runs in $\mathsf{poly}(n)$, hence $\ell = \mathsf{poly}(n)$
2: **for** `each` $i \in [\ell]$ **do**
3: If $\mathcal{A}(x, y_i)$ accepts, ACCEPT.
4: **end for**
5: REJECT.

Indeed, if the answer if NO, then $\Gamma_x = 0$, so the algorithm reaches step 5 and rejects. If the answer is YES, then $|\Gamma_x| \geq \frac{1}{2}$ and since H_r is a $\frac{1}{2}$-hitting set, we have that $H_r \cap \Gamma_x \neq \phi$. Let us say $y_i \in \Gamma_x \cap H_r$. Hence, $\mathcal{A}(x, y_i)$ accepts. Thus, $\mathcal{A}'(x)$ accepts. The algorithm is deterministic and runs in polynial time. □

3.6 Connections between various objects

In the past few lectures, we informally introduced several mathematical objects in relation to our main task - derandomizing randomized algorithms. We also stated that they all have the following common feature. *By non-constructive arugments, we can prove that these objects exists for the range of paramters we want, but the question that remains is whether they can be constructed explicitly or not.*

It turns out that there are lots of interconnections between these objects (although sometimes for weaker set of parameters). The remaining part of the course also will demonstrate these connections, in addition to detailing out the applications of these objects in connection to the derandomization task.



In the next lecture, we will first introduce the tools to show non-constructive existence and demonstrate them with various examples.

Proving the Existence : The Probabilistic Method

Lecturer : Jayalal Sarma

WEEK

4

The probabilistic method is a simple and powerful technique to show that some combinatorial object with certain properties exists. The idea is quite simple, design a random experiment to obtain the combinatorial object and then show that the probability that the properties does not get satisfied with probability strictly less than 1. Hence, by probability arguments, there must exist the combinatorial object having the properties in the underlying sample space.

4.1 Hypergraph 2-coloring

Now we show the first application of the probablistic method through the example of hypergraph 2-coloring. A hypergraph is a pair of two sets (V, E) referred to as vertices and edges (or hyperedges).The edges are subsets of the vertices. If all the sets in E have size k, then we call it the k-uniform hypergraph. Notice that a graph is a 2-uniform hypergraph. A proper (vertex) coloring is an assignments of colors to the vertices of a hypergraph so that no edge is monochromatic. Note that this naturally generalizes graph (vertex) coloring where it is insisted that adjacent vertices (that is, the elements of the edges) get different colors.

We show the following theorem using probabilistic method.

THEOREM 4.1.1 **(Erdös (1963))**. *If $H(V, E)$ is a k-uniform hypergraph with less than 2^{k-1} hyperedges then there exists a proper 2-coloring of H.*

Proof. Let $H(V, E)$ be a hypergraph such that $|E| < 2^{k-1}$. The experiment we set up is to color each vertex in the graph with one of the two colors (say red and blue) uniformly at random (that is, with probability $\frac{1}{2}$ the vertex will be colored red and with probability $\frac{1}{2}$ it will be colored blue). Let A_e be the event that all the k vertices in the hyperedge $e \in E$ gets the same color. We calculate $Pr[A_e]$ first. Since the monochromatic color for A_e can be chosen in two ways:

$$Pr[A_e] = \frac{2}{2^k} = \frac{1}{2^{k-1}}$$

The coloring is not proper if the event A_e happens for at least one of the hyperedge e. Hence,

$$Pr[\text{ coloring is not proper }] \leq Pr\left[\bigcup_{e \in E} A_e\right] \leq \sum_{e \in E} Pr[A_e] \leq \frac{|E|}{2^{k-1}} < 1$$

That is, if we choose a random 2-coloring, we will get a proper-coloring with probability greater than zero. This, in particular implies that there exists a proper 2-coloring of the hyper-

graph. □

We remark that by simply restricting $|E| < 2^{k-2}$ we could have proved that if we choose a random 2-coloring, we will get a proper-coloring with probability greater than $\frac{1}{2}$. However, this implies that there is even a randomized algorithm to construct the coloring, in this case. Indeed, if we can efficiently derandomize the algorithm, it makes the proof constructive.

The following exercise tells us that the above scheme can also be applied for higher number of colors too.

Exercise 4.1.2. Suppose $k > 2$ and let H be a k-uniform hypergraph with 4^{k-1} edges. Show that there is a 4-colouring of $V(H)$ such that no edge is monochromatic.

Curiosity 4.1.3 (Property-B Conjecture). A hypergraph H has **Property B** (or 2-colorable) if there is a red-blue vertex-coloring with no monochromatic edge. A hypergraph with property B is also called bipartite, by analogy to the bipartite graphs. Erdos (1963) asked: What is the minimum number of edges $m_2(k)$ of a k-uniform hypergraph not having property B? Indeed, the above discussion implies that $m_2(k) \geq 2^{k-1}$. Erdos proved an upper bound of $m_2(k) \leq O(k^2 2^k)$. The best known bounds are :

$$\Omega\left(\sqrt{\frac{k}{\ln k}}2^k\right) \leq m_2(k) \leq O\left(k^2 2^k\right)$$

The upper bound is due to Erdos (1964) and the lower bound went through a series of improvements to reach the above bound Radhakrishnan and Srinivasan (2000).

$$\left\{\begin{array}{c} m_2(k) \geq \left(\frac{1}{2}\right) 2^k \\ \text{[Erdos, 1963]} \end{array}\right\} \rightarrow \left\{\begin{array}{c} m_2(k) \geq \left(\frac{k}{k+4}\right) 2^k \\ \text{[Schmidt, 1964]} \end{array}\right\} \rightarrow \left\{\begin{array}{c} m_2(k) \geq \left(\sqrt[3]{k}\right) 2^k \\ \text{[Beck, 1978]} \end{array}\right\} \rightarrow \left\{\begin{array}{c} m_2(k) \geq \left(\sqrt{\frac{k}{\ln k}}\right) 2^k \\ \text{[RS, 2000)]} \end{array}\right\}$$

It is believed that $k2^k$ is the right asymptotic bound for $m_2(k)$. In fact, this is a conjecture due to Erdos and Lovasz: $m_2(k) \in \Theta(k2^k)$.

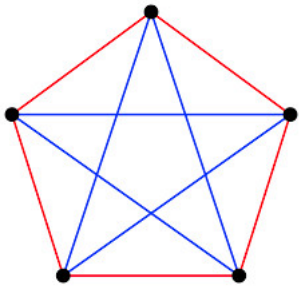
4.2 Diagonal Ramsey Number Bound

We now show the original application of the probabilistic method, when it was introduced by Erdös. This is to prove a lower bound for certain ramsey numbers. We now quickly introduce Ramsey numbers in this lecture.

The standard starting point is the following brain teaser : any party with at least 6 people will contain a group of three mutual friends or a group of three mutual non-friends. Indeed, the immediate combinatorial argument goes like this : call the people $1, 2, 3, 4, 5, 6$. Either 1 has three friends or three non-friends. Without loss of generality, suppose that $2, 3, 4$ are all friends with 1. Then if any pair of them are friends with each other, that pair plus 1 forms a group of three mutual friends. If no two of them are friends, then they are a group of three mutual non-friends.

Note that the above can also be done graph theoretically where we consider a 6 vertex graph with drawing an edge between two vertices $i, j \in \{1, 2, \ldots 6\}$ if and only if they are friends with each other. Now the above argument can be tranlated to graph : *any graph on 6 vertices will contain either a clique on 3 vertices or an independent set on 3 vertices.*

An alternate way to represent the above problem is by 2-coloring the edges of the complete graph K_6, by red if the two vertices are friends with each other and with blue otherwise. Now the above statement becomes : *in any 2-coloring of K_n, there is a monochromatic triangle* - which indeed, is more concise statement.



Is there a peculiarity with 6 and can the same be argued for 5? It turns out that we cannot and there is the following counter example (which we represent by the 2-coloring of K_5). Thus 6 is the minimum number such that for any 2-coloring of the edges of K_6 there will exist a monochromatic triangle.

The Ramsey theory asks a general extremal combinatorics question of this form : For any $s, t \in \mathbb{N}$, what the minimum number n such that there is a guarantee of the form : any 2-coloring of the edges of the graph G has either a red K_s or a blue K_t in it. This number exists (as proved by Frank Ramsey) and is called the Ramsey Number $R(s,t)$. In this language, the above argument says $R(3,3) = 6$. In fact, the existance argument for Ramsey number actually gives the followng upper bound.

PROPOSITION 4.2.1. $R(s,t) \leq R(s,t-1) + R(s-1,t)$.

Computing other Ramsey numbers has attracted a lot of attention from combinatorialists. However, we know very little still. $R(s,2) = s$, $43 \leq R(5,5) \leq 49$ etc. The numbers where $s = t$ are the diagonal entries of the Ramsey matrix (which is natural to imagine given the above). By applyng the above theorem, we have that:

$$R(s,s) \leq 2^{2s}\sqrt{s}$$

In the rest of this section, we will concetrate on lower bounds for the diagonal Ramsey numbers. Notice that to show lower bound $R(s,s) \geq n$, we need to show that there is a 2-coloring of K_n where there is no monochromatic K_s. A constructive lower bound of this kind was discovered by Nagy which shows :

$$R(s,s) \geq \binom{s}{3}$$

We now apply probabilistic method in order to obtain a stronger lower bound. Erdos, in 1947, introduced probabilistic methods in his paper *Some Remarks on the Theory of Graphs* for proving this lower bound.

THEOREM 4.2.2. *The diagonal Ramsey number $R(s,s)$ is at least $\lfloor 2^{\frac{s}{2}} \rfloor$. Equivalently, when $n = 2^{\frac{s}{2}}$, there exists a 2-coloring of K_n where there is no monochromatic K_s.*

Proof. As usual, we need to show the existance of a 2-coloring to edges. We set up the following experiment. Color each edge uniformly at random with red or blue. The total number of possible colorings is $2^{\binom{n}{2}}$. The probability of any particular color configuraton is exactly $\frac{1}{2^{\binom{n}{2}}}$.

We need to prove an upper bound of the bad events. Let $S \subseteq n$ of size s. Our coloring is bad if

S is colored monochromatic under the above coloring.

$$Pr[S \text{ is monochromatic}] \leq \frac{2 \times 2^{\binom{n}{2}-\binom{s}{2}}}{2^{\binom{n}{2}}} \leq 2^{1-\binom{s}{2}}$$

$$Pr[\exists S : S \text{ is monochromatic}] \leq \sum_{S \subseteq [n]:|S|=s} Pr[S \text{ is monochromatic}] \leq \binom{n}{s} 2^{1-\binom{s}{2}}$$

If we show that $\binom{n}{s}2^{1-\binom{s}{2}}$ is less than 1 for $n = 2^{s/2}$, then we are done by probablistic argument.

$$\binom{n}{s} 2^{1-\binom{s}{2}} \leq \frac{n^s}{s!} 2^{1-(s/2)+(s^2/s)} \leq \frac{n^s}{2^{s^2/2}} \frac{2^{1+s/2}}{s!} < 1$$

Hence the proof. □

Exercise 4.2.3 (See Problem Set 2 (Problem 1))**.** A tournament is a directed graph $G(V, E)$ on n vertices where for every pair (i, j), there is either an edge from i to j or from j to i, but not both (it represents real tournaments, where we interpret (i, j) directed edge as player i beats player j. There is no draw and all pairs of players play a game with each other).

A tournament T is said to have **k-championship property** if for any set of k vertices in the tournament, there is some vertex in V that has a directed edge to each of those k vertices.

Can k-Championship property occur in small tournament graphs? For example, for $k = 1$, a tournament will need at least 3 vertices to have the k-Championship property. If $k = 2$, a tournament will need at least 5 vertices to have k-Championship property.

Show that there are tournments of size $O(k^2 2^k)$ having k-Championship property. [Hint : Consider a random tournament. Fix a set S of k vertices and some vertex $v \notin S$. What is the probability that v is the champion in S?]

Exercise 4.2.4 (See Problem Set 2 (Problem 2)**.** Let $G(V, E)$ be a graph. A set of vertices $D \subseteq V$ is called dominating with respect to G if every vertex in $V \setminus D$ is adjacent to a vertex in D. $\delta(G)$, the minimum degree amongst G's vertices, is strictly positive. Then G contains a dominating set of size less than or equal to:

$$\frac{n(1 + \log(1 + \delta))}{1 + \delta}$$

[Hint : Choose a subset $X \subseteq V$ at random (with each vertex in with probability p). Let $Y \subseteq V \setminus X$ having no neighbor in X. Estimate $X \cup Y$.]

4.3 Expectation Method

We now apply the method of expectation with probablistic method. The basic idea is that when we want to show the existence of an object for which a parameter (say graph with at least certain number of connected components) satisfy some lower/upper bounds. We first define the parameter as the random variable and show that the expected value satisfies the bounds which implies that there exists an object which satisfies the bounds. The following lemma makes this more precise.

LEMMA 4.3.1 (**Expectation Method**). *For any random variable X,*

$$\mathbb{E}[X] \geq t \Rightarrow Pr[X \geq t] > 0$$

Proof. Assume for the sake of contradiction that $\Pr[X \geq t] = 0$.

$$\mathbb{E}[X] = \sum_{w \in \Omega} X(w)P(w)$$

If $\Pr[X \geq t]$ was zero, then by definition,

$$\sum_{\substack{w \in \Omega \\ X(w) \geq t}} P(w) = 0$$

Since $P(w)$ is non-negative, we have that, for every $w \in \Omega$, with $X(w) \geq t$, $P(w) = 0$.

$$\mathbb{E}[X] = \sum_{w \in \Omega} X(w)P(w) = \sum_{\substack{w \in \Omega \\ X(w) < t}} X(w)P(w) < t \left(\sum_{w \in \Omega} P(w) \right) = t$$

That will be a contradiction.

□

Question to students : Does the same statement holds when $\geq$ is replaced by any of $\leq, <$ or$>$?

4.3.1 Sum-free Sets

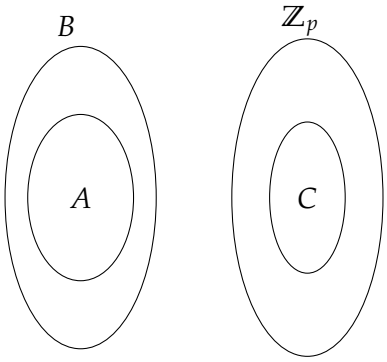
To apply the expectation method, we need to make a suitable choice of the random variable X so that it is not difficult to compute its expected value. We demonstrate thetechnique by using the example of sum-free sets. We start witht he definitions.

A subset A of positive integers is called *sum-free*, if $\nexists x, y, z \in S$ such that $x + y = z$ (the number may repeat). Given a set of positive integers, say $B = \{b_1, b_2, \ldots b_n\}$ how large a sum-free subset is it guaranteed to contain? A simple example to try out is $B = \{1, 2, \ldots n\}$. If we choose all odd numbers in the set, or even the elements more than $\frac{n}{2}$, they all are sumfree subsets of size at least $\frac{n}{2}$. Erdös answered this question using a cute mathematical statement.

THEOREM 4.3.2. *For any set $B = \{b_1, b_2, \ldots b_n\}$ of positive integers, there must exist a sumfree subset $A \subseteq B$ such that $|A| > \frac{n}{3}$.*

Proof. We will follow the above outlined strategy to define an appropriate random variable but in an indirect way. We choose a prime $p > 2b_n$ where b_n is the largest number in B and that $p = 3k + 2$. Now all addition operation between the elements in B are faithfully captured in the the set $\mathbb{Z}_p$ where addition is done modulo p. That is there is a natural mapping from B to $\mathbb{Z}_p$ which preserves the sum operations. The mapping can be thought of as the identity function since $p \geq 2b_n$. Indeed, $\phi(b_i + b_j) = \phi_(b_i) + \phi(b_j)$ from the definitions itself.

Sum-free sets in $\mathbb{Z}_p$: Instead of showing a sum-free subset of B, we show a sum-free subset C of $\mathbb{Z}_p$ first. We explicitly define C first. Consider the middle third elements of $\mathbb{Z}_p$. That is, $C = \{k+1, k+2, \ldots 2k+1\}$. And $|C| > \frac{p-1}{3}$. We claim that C is sum-free. To see this : $x, y, z \in C$, $x + y \geq 2k+2$ even for the smallest elements in C, and $x + y = 4k+2$ (which is at most k modulo $3k+2$) even for the largest elements in C. Hence none of these sums will land back in C as a result of the modulo operation. Hence C is sum-free.



Pull-back to B - A failed attempt: Now how do we go back to get a subset of B which is sumfree. A natural idea is to use the "pull back" of the function ϕ. The pre-image of C, namely $\phi^{-1}(C)$ must be sumfree as well. To see this : if there exists $x, y, z \in \phi^{-1}(C)$ such that $x + y = z$, then $\phi(x) + \phi(y) = \phi(z)$. Since $\phi(x), \phi(y), \phi(z) \in C$ by definition, this implies that C is not sumfree and that is a contradiction. But then, why should the size of the pre-image by at least $\frac{n}{3}$? It could even be that none of the elements in B are mapped to C, so the pre-image can even be empty !.

Way forward - random scaling: Since the above mapping may not be good, we consider variants of ϕ which also preserves the addition operation in B. Consider any $\alpha \in \mathbb{Z}_p \setminus \{0\}$, the map $\phi_\alpha : B \to \mathbb{Z}_p$:

$$\phi_\alpha : b_i \mapsto \alpha b_i \mod p$$

Let us quickly check if this is a function which respects the sum operation: if $b_i + b_j = b_k$, $\phi_\alpha(b_i) + \phi_\alpha(b_j) = \alpha b_i \mod p + \alpha b_j \mod p = (\alpha b_i + \alpha b_j) \mod p = \phi_\alpha(b_i + b_j) = \phi_\alpha(b_k)$ As a consequence, the pre-image of C under any α, $\phi_\alpha^{-1}(C)$ will be sumfree.
Now comes the punch line argument. We will argue :

$$\exists \alpha \in \mathbb{Z}_p \setminus \{0\} \textit{ such that } \phi_\alpha^{-1}(C) \textit{ is a large subset of } B.$$

It is to prove this last statement that we apply the expectation method described above. To begin with, let us write the claim more precisely:

CLAIM 4.3.3. *$\exists \alpha \in \mathbb{Z}_p \setminus \{0\}$ such that $|\phi_\alpha^{-1}(C)| > n/3$.*

Original statement that we wanted to prove said, $\exists A \subseteq B$, now it says $\exists \alpha \in \mathbb{Z}_p \setminus \{0\}$. Let us set up the experiment. Choose α uniformly at random from $\mathbb{Z}_p \setminus \{0\}$. Let us defined the random variable X as:

$$X = |\phi_\alpha^{-1}(C)| = |\{i \in [n] \mid \phi_\alpha(b_i) \in C\}|$$

As per expectation method, we just need to prove that $\mathbb{E}[X] > \frac{n}{3}$. Let us breakdown this random variable into simpler ones in terms of the individual b_is. Let us define the indicator random variable corresponding to the event $\phi_\alpha(b_i) \in C$.

$$X_i^\alpha = \begin{cases} 1 & \text{if } \phi_\alpha(b_i) \in C \\ 0 & \text{otherwise} \end{cases}$$

Note that, $X = \sum_{i=1}^n X_i$. Hence, by linearity of expectation, $\mathbb{E}[X] = \sum_{i=1}^n \mathbb{E}[X_i^\alpha]$. Thus, we need to

lower bound $\mathbb{E}[X_i^\alpha]$. Since it is an indicator random variable, $\mathbb{E}[X_i^\alpha] = \Pr_\alpha[\phi_\alpha(b_i) \in C]$ which we will compute now.

For a b_i and a a target element on RHS (consider a non-zero element on RHS $y \in \mathbb{Z}_p \setminus \{0\}$) how many α can map b_i to y? This is exactly one element $\alpha = b_i y^{-1} \mod p$. Thus, if we choose an α uniformly at random, probability that it maps b_i to y is $\frac{1}{p-1}$. If, in addition, this y has to be in the set C, then there are exactly $|C|$ choices of α which sends b_i to C. That is:

$$\mathbb{E}[X_i^\alpha] = \Pr_\alpha[\phi_\alpha(b_i) \in C] = \frac{|C|}{p-1} > \frac{1}{3}$$

Thus, as planned, we have shown that $\mathbb{E}[X] = \sum_{i=1}^n \mathbb{E}[X_i^\alpha]$ is greater than $\frac{n}{3}$. Hence as per Lemma 4.3.1 (expectation method), $\Pr_\alpha[X \geq \frac{n}{3}] = \Pr_\alpha\left[|\phi_\alpha^{-1}(C)| > \frac{n}{3}\right] > 0$. Hence, by probabilistic method, we have that there exists an α such that $|\phi_\alpha^{-1}(C)| > \frac{n}{3}$. Hence there exists an A (namely $\phi_\alpha^{-1}(C)$) which is a subset of B and is sum-free. This completes the argument. □

4.3.2 Crossing Number Lower Bound

Now we discuss a totally different application of expectation which is not necessarily an incarnation of probabilistic method, but still is a fun example. We set up the context first.

Planar graphs are graphs which can be embedded on the plan without any crossing of the edges. Indeed, there are non-planar graphs. How do we measure non-planarity? There are many ways - one of them is the crossing number of the graph. The crossing number $cr(G)$ of a graph G is the least number of crossings in any embedding of the graph on the plane. Indeed, if the graph G is planar, then $cr(G) = 0$. As an exercise, check out the crossing number of the non-planar graphs $K_{3,3}$ and K_5. Have fun !.

What can be the bounds on crossing number in terms of the edges and vertices of the graph? Clearly, if the graph is highly dense, then we do expect the crossing number to be high. Hence one should expect a lower bound in terms of the number of edges.

We quickly review the Euler's formula for planar graphs which can also be used to bound the crossing number.

PROPOSITION 4.3.4 (**Euler's Formula for Planar Graphs**). *Any planar graph G, with n vertices, m edges and f faces, and c components, must satisfy:*

$$n - m + f = c + 1 \tag{4.11}$$

Proof. The proof is an induction on m. If there are no edges ($m = 0$), then $f = 1$, $c = n$. The formula gets satisfied. Let $m > 0$, and assume that the statement is true for graphs with less than m edges. Consider an edge $e = (u, v)$. Remove this edge from G to obtain a new graph G'. Let $c' \geq c$ be the number of components in G'. We have two cases:

Case 1: $c' > c$ **:** The number of components can increase at most bt 1. $c' = c + 1$. By removing the edge, the number of faces does not change since the removed edge would have been the boundary for the outer face of the embedding. Hence $m' = m - 1$, $f' = f$. Hence the equation 4.11 will still be satisfied.

Case 2: $c' = c$ **:** By removing the edge, we would have decreased the number of faces by 1. Hence $m' = m - 1$, $f' = f - 1$. Hence the equation 4.11 will still be satisfied.

□

COROLLARY 4.3.5. *Any connected planar graph G on $n \geq 4$ vertices cannot have more than $3n - 6$ edges.*

Proof. Let the number of edges in the boundary of the i^{th} face be m_i. Observe that each edge can be the boundary of at most two faces. This gives: $\sum_{i=1}^{f} m_i \leq 2m$. Since $m_i \geq 3$ (even for outer face, since G is connected), this gives $f \leq \frac{2m}{3}$. Applying this relation to Euler's formula gives, $m = n + f - 2 \leq n + \frac{2m}{3} - 2$. This gives that $m \leq 3n - 6$. □

The next corollary is about a lower bound on the crossing number.

COROLLARY 4.3.6. *For any graph with n vertices and m edges, we have that*

$$cr(G) \geq m - 3n + 6$$

Proof. The idea is quite simple. Take the best embedding of the graph which achieves the crossing number. For each crossing put in a vertex at the geometric point of the crossing edges. What we get is a planar embedding of a new planar graph G'. Indeed the number of vertices have increased by $cr(G)$. The number edges have increased by $2cr(G)$. $n' = n + cr(G)$ and $m' = m + 2cr(G)$. Applying the above corollary to the graph G':

$$m + 2cr(G) \leq 3(n + cr(G) - 6$$

This gives that, $cr(G) \geq m - 3n + 6$, thus completing the proof. □

Thus we have one lower bound for crossing number. We will use our expectation method to derive a much stronger lower bound.

THEOREM 4.3.7 ([Beck, 1978]). *Let G be a simple graph with $m > 4n$. Then,*

$$cr(G) \geq \frac{1}{64}\frac{m^3}{n^2}$$

Proof. Again, our aim is to define an appropriate experiment and a random variable whose expectation will be lower bounded. Let G be the graph, and let $\tilde{G}$ be the embedding of G on the plane which achieves the crossing number $cr(G)$. That is, the emebedding witnesses $cr(G)$ crossings.

We design the following experiment. Choose a subset of vertices V by choosing each vertex of G independently with probability $p = \frac{4n}{m}$. Let H be the induced subgraph on these vertices and let $\tilde{H}$ be the embedding of H derived from $\tilde{G}$. Notice that H may have a better embedding which achieves a lower crossing number than what $\tilde{H}$ witnesses.

The crossing number of H, $cr(H)$ is a random variable. In addition, the number of vertices, number of edges, and the number of crossings of $\tilde{H}$ are all random variable. Let X, Y, Z be these random variables respectively.

Since $\tilde{H}$ is only one of the possible embeddings of H, $Z \geq cr(H)$. Since the number of vertices is independent of the embedding, by applying Corollary 4.3.6, we have that $cr(H) \geq Y - 3X$ (we

are ignoring the additive constant 6 but the lower bound is still valid), Thus we have that :

$$Z \geq Y - 3X \text{ which implies that } \mathbb{E}[Z] \geq \mathbb{E}[Y] - 3\mathbb{E}[X]$$

Now we need to compute each of them. Since we are choosing each vertex to be in H indendently with probability p, $\mathbb{E}[X] = pn$. Since for an edge from G to be chosen, both of its end points must be chosen, $\mathbb{E}[Y] = p^2 m$. For a crossing in $\tilde{G}$ to appear in $\tilde{H}$, all the 4 vertices involved in the crossing must be chosen. Hence $\mathbb{E}[Z] = p^4 cr(G)$. Plugging this in yields that:

$$p^4 cr(G) \geq p^2 m - 3pn \Rightarrow cr(G) \geq \frac{pm - 3n}{p^3} = \frac{n}{(4n/m)^3} = \frac{1}{64}\frac{m^3}{n^2}$$

the required lower bound. □

Exercise 4.3.8 (See Problem Set 2(Problem 3))**.** Use expectation method to show that every graph having a matching of size m has a bipartite subgraph with at least $\frac{1}{2}(|E(G)| + m)$ edges. (Hint: how can we choose a random bipartition such that the edges of the matching have their endvertices in opposite parts?)

Exercise 4.3.9 (See Problem Set 2(Problem 4))**.** Let $t, \ell, d \in \mathbb{N}$ and $t \leq \ell \leq d$. A family of sets $S_1, S_2, \ldots S_m \subseteq [d]$ is said to be a (d, ℓ, t)-design if:

- $\forall i \in [m]$, $|S_i| = \ell$.
- For all $i, j \in [m], i \neq j$, $|S_i \cap S_j| < t$.

That is, the family is ℓ-uniform (each set of size ℓ) subsets of $[d]$ with intersection sizes at most t. We will have an application soon in the course for such sets. Given ℓ, we would want maximize the number of sets that can be "packed" in - that is maximise m, while keeping t and d to be small. But do such sets exist for all parameters? A typical question that we face in this course, and we use our tools:

(a) Prove that if, $m\binom{\ell}{t}^2 < \binom{d}{t}$, then there exists an (d, ℓ, t)-design $S_1, S_2, \ldots S_m \subseteq [d]$.
Hint : If the sets are chosen at random, then prove that for every $S_1, S_2, \ldots S_{i-1}$:

$$\mathbb{E}_{S_i}\left[\#j < i : |S_i \cap S_j| \geq t\right] < 1$$

(b) Use this to conclude that for every constant $c > 0$, $\ell, m \in \mathbb{N}$, there exists and (d, ℓ, t)-design, $S_1, S_2, \ldots S_m \subseteq [d]$ with $d = O\left(\frac{\ell^2}{t}\right)$ and $t = c \log m$. In particular, setting $m = 2^\ell$, we can fit exponentially many sets of size ℓ in a universe of size $d = O(\ell)$ while keeping the intersections as logarithmically small.

(c) Use method of conditional expectations to derandomize the above to show how to construct designs as in Parts 1 and 2 deterministically in time $\mathsf{poly}(m, d)$.

4.4 Refined Probablistic Method : Lovaśz Local Lemma

In many situations where we apply the probabilistic method, one is trying to show that it is possible to avoid *bad events* $\mathcal{E}_1, \mathcal{E}_2, \ldots \mathcal{E}_n$ with positive probability. That is, we show that, $\Pr[\bigcap_i \overline{\mathcal{E}_i}] > 0$ where $\overline{\mathcal{E}_i}$ denotes the complementary event to $\mathcal{E}$.

One way to show $\Pr[\bigcap_i \overline{\mathcal{E}_i}] > 0$ is to use $\Pr[\bigcap_i \overline{\mathcal{E}_i}] = 1 - \Pr[\bigcup_i \mathcal{E}_i]$. We now use the fact that $\Pr[\bigcup_i \mathcal{E}_i] \leq \sum_i \Pr[\mathcal{E}_i]$. We try to show that this last sums up to less than 1. Indeed the last inequality is weak. That is, $\sum_i \Pr[\mathcal{E}_i]$ may even be more than 1, but still $\Pr[\bigcup_i \mathcal{E}_i] < 1$.

If in addition, we knew that the events $\rangle$ are independent, then it suffices to have that $\Pr[\mathcal{E}_i] < 1$ for each i. Then,

$$\Pr[\bigcap_i \overline{\mathcal{E}_i}] = \prod_i (1 - \Pr[\mathcal{E}_i]) > 0$$

Indeed, it might tougher to design independent events whose simultaneous avoidance is enough to estalish the existance of the object that we are looking for.

Hence, we need a framework, where even if $\mathcal{E}_i$s are dependent, still one can estimate and argue that $\Pr[\bigcap_i \overline{\mathcal{E}_i}]$ is less than 1. This is what Lovász Local Lemma helps us do, where the dependence is a little more structured.

4.4.1 Dependency Graph

As mentioned above, Lovász Local Lemma helps us to handle the situations where there is only local dependancy. The structure of the dependency between the events under consideration is represented by the dependency graph.

To define the dependency graph, we need the notion of mutual independence which we defined earlier.

DEFINITION 4.4.1. *Let $\mathcal{E}_1, \mathcal{E}_2, \ldots \mathcal{E}_n$ be events. Let $J \subseteq [n]$. We say that event $\mathcal{E}_i$ is mutually independent of the events $\{\mathcal{E}_j \mid j \in J$ for all $J_1, J_2 \subseteq J$ such that $J_1 \cap J_2 = 0$*

$$\Pr[\mathcal{E}_i \cap \left(\cap_{j \in J_1} B_j\right) \cap \left(\cap_{j \in J_2} \overline{B_j}\right)] = \Pr(\mathcal{E}_i) \Pr[\left(\cap_{j \in J_1} B_j\right) \cap \left(\cap_{j \in J_2} \overline{B_j}\right)]$$

Now we are ready to represent the dependency information in the form of a directed graph. The vertices of the graph are the events $\{1, 2, \ldots n\}$.

DEFINITION 4.4.2 (**Dependency Graph**). *A (directed) graph $G([n], E)$ is a dependency graph on events $\mathcal{E}_1, \mathcal{E}_2, \ldots \mathcal{E}_n$ if each event $\mathcal{E}_i$ is mutually independent of its non-neighbors $\{\mathcal{E}_j \mid (i, j) \notin E\}$.*

REMARK 4.4.3. Notice that for the same set of events, there can be several dependency graphs. For example, if a directed graph G is a dependency graph for a set of events. If we add more edges to G, it will still be a dependency graph for the same set of events.

4.4.2 The Symmetric LLL and an Application

A simpler version to study first is the symmetric version. In fact, in this version we do not even use the structure of the graph, but only the degree bounds. We first state the lemma:

LEMMA 4.4.4. *Suppose $p \in (0,1)$, $d \geq 1$, and $\mathcal{E}_1, \mathcal{E}_2, \ldots \mathcal{E}_n$ are events such that $\Pr[\mathcal{E}_i] \leq p$ for all i. If each E_i is mutually independent of all but d other events*[11] *and $ep(d+1) \leq 1$, then: $\Pr[\bigcap_i \overline{\mathcal{E}_i}] > 0$*

We will prove the symmetric version in the next section. But before that we apply it. We apply to it the hypergraph 2-coloring problem that we talked as the first example for probablistic method.

Application to Hypergraph 2-coloring: We demonstrate that, we can still prove a k-uniform hypergraph is 2-colorable even if it has more than 2^{k-1} edges (a condition which we required to establish 2-colorability using Probablistic method) if there is some structural property for the edges : *every edge intersects at most d other edges*.

To apply the lemma, let us think of events $\mathcal{E}_i$ in the same way : whether the i^{th} edge e_i is monochromatically colored or not. As we did in that example, we have that

$$\Pr[\mathcal{E}_i] = \frac{2}{2^k} = \frac{1}{2^{k-1}} = p$$

We now claim that the dependency condition is satisfied. Clearly for a vertex which corresponds to the event of monochromatic coloring of the set $\mathcal{E}_i$, the other events which does not even have a common vertex with $\mathcal{E}_j$ are clearly independent of the event $\mathcal{E}_i$. Hence each event is dependent on at most d other events and hence the degree is upper bounded by d. We can always add additional edges to make the graph d-regular. The last condition that needs to be checked is whether the individual event probability is low enough.

$$ep(d+1) = e(d+1)\frac{1}{2^{k-1}} \leq 1 \text{as required by the lemma, if we ensure that } d < \frac{2^{k-1}}{e} - 1$$

This gives the following theorem.

THEOREM 4.4.5. *For any k-uniform hypergraph with the guarantee that every hyperedge intersects with at most 2^{k-3} other hyperedges, then the hypergraph is 2-colorable.*

Exercise 4.4.6. For a set of Boolean variables $x_1, x_2, \ldots x_n$, a k-CNF formula ϕ has the form $\phi = C_1 \wedge \ldots \wedge C_m$. Each clause C_j is an or of some set of k literals, where each literal is either x_i or x_i for some $i \in [n]$. Clauses C_j and C_k are said to intersect if $\exists x_i$ such that both clauses contain either x_i or $\neg x_i$. A satisfying assignment is a setting of the x_is that makes ϕ evaluate to true. Deciding the existence of a satisfying assignment for a Boolean CNF formula is another well-studied (and NP-complete) problem. This problem asks you to show that a satisfying assignment exists in certain cases. If each clause intersects at most $\frac{2^k}{e} - 1$ other clauses, then show that ϕ is satisfiable. (Hint: Use LLL).

4.4.3 The Asymmetric LLL and an Application

We state and prove the general version of LLL and prove it first. We will also show an application of the lemma.

[11] This implies that there is a regular dependency graph

THEOREM 4.4.7 (**Lovász Local Lemma**). *Suppose $\mathcal{E}_1, \mathcal{E}_2, \ldots \mathcal{E}_n$ are n events with G as a depedency graph. If there exists $x_1, x_2, \ldots x_n \in [0,1)$ such that for every $i \in [n]$:*

$$\Pr[\mathcal{E}_i] \leq x_i \prod_{(i,j)\in E} (1 - x_j) \tag{4.12}$$

then,

$$\Pr\left[\bigcap_{i=1}^{n} \overline{\mathcal{E}_i}\right] \geq \prod_{i=1}^{n} (1 - x_i) > 0$$

An important example is when the events are actually independent. In this case the LLL result is tight. Also, while trying to apply LLL to a situation, having designed the set of events we have the design freedom for the depedency graph and the values $x_1, x_2, \ldots x_n \in (0,1)$.

Deriving the Symmetric LLL: The dependency graph is d-regular. Choose $x_i = 1/(d+1)$. With this, we want to show that the premise of the LLL (Equation 4.12) is satisfied. Indeed, the RHS is $(d+1)(1 - \frac{1}{d+1})^d \geq \frac{1}{(d+1)e} \geq p \geq \Pr[\mathcal{E}_i]$. Hence the symmetric version follows.

Proof of the Asymmetric LLL: Staring at $\Pr\left[\bigcap_{i=1}^{n} \overline{\mathcal{E}_i}\right]$ indicates that may be one can grow the probability bound starting from thinner intersections. To do this, we will define, for $S \subseteq [n]$,

$$\mathsf{P}(S) = \Pr\left[\bigcap_{i\in S} \overline{\mathcal{E}_i}\right]$$

Clearly, on one side, we have that $P(\phi) = 1$, and on the other side we want to derive $P([n])$. This calls for a proof by induction on the size of $|S|$. Thus we strengthen the statement to the following lemma which we prove by induction. Note that the following lemma implies the LLL.

LEMMA 4.4.8. *Let $S \subseteq [n]$. For any $i \in S$:*

$$\mathsf{P}(S) \geq \mathsf{P}(S \setminus \{i\})(1 - x_i)$$

Proof. Base case is trivial (figure this out !). We will assume the statement holds true whenever $|S| \leq k$ and then we will prove it for $|S| = k+1$. Let S be a subset of $[n]$ of size $k+1$ and fix $i \in S$. We need to appeal to indepedence of the events in S for which we need the neighborhood.

$$\Gamma(i) = \{j \mid (i,j) \in E\} \quad \text{and} \quad \Gamma^+(i) = \{i\} \cup \Gamma(i)$$

We need to estimate a lower bound for $P(S)$. We use the fact that for any event A, $\Pr[A] =$

$\Pr[A \cap B] + \Pr[A \cap \overline{B}]$. Use $A = \bigcap_{i \in S \setminus \{i\}} \overline{\mathcal{E}_i}$ and $B = \mathcal{E}_i$. This gives:

$$\begin{aligned}
\mathsf{P}(S) = \Pr\left[\bigcap_{i \in S} \overline{\mathcal{E}_i}\right] &= \Pr[A] - \Pr[A \cap B] \\
&= \Pr\left[\bigcap_{i \in S \setminus \{i\}} \overline{\mathcal{E}_i}\right] - \Pr\left[\mathcal{E}_i \cap \bigcap_{i \in S \setminus \{i\}} \overline{\mathcal{E}_i}\right] \\
&\geq \Pr\left[\bigcap_{i \in S \setminus \{i\}} \overline{\mathcal{E}_i}\right] - \Pr\left[\mathcal{E}_i \cap \bigcap_{i \in S \setminus \{\Gamma^+(i)\}} \overline{\mathcal{E}_i}\right] \\
&= \Pr\left[\bigcap_{i \in S \setminus \{i\}} \overline{\mathcal{E}_i}\right] - \Pr[\mathcal{E}_i] \Pr\left[\bigcap_{i \in S \setminus \{\Gamma^+(i)\}} \overline{\mathcal{E}_i}\right] \\
&= \mathsf{P}[S \setminus \{i\}] - \Pr[\mathcal{E}_i]\mathsf{P}[S \setminus \Gamma^+(i)]
\end{aligned}$$

Dividing by $\mathsf{P}[S \setminus \{i\}]$,

$$\frac{\mathsf{P}(S)}{\mathsf{P}[S \setminus \{i\}]} = 1 - \Pr[\mathcal{E}_i]\frac{\mathsf{P}[S \setminus \Gamma^+(i)]}{\mathsf{P}[S \setminus \{i\}]}$$

Since we are only involving elements from S, we can trstrict our attention to the neighbors who are in S. Let $\Gamma(i) \cap S = \{b_1, b_2, \ldots b_d\}$. Now the second term can be simplified.

$$\frac{\mathsf{P}[S \setminus \Gamma^+(i)]}{\mathsf{P}[S \setminus \{i\}]} = \frac{\mathsf{P}[S \setminus \{i, b_1\}]}{\mathsf{P}[S \setminus \{i\}]} \times \frac{\mathsf{P}[S \setminus \{i, b_1, b_2\}]}{\mathsf{P}[S \setminus \{i, b_1\}]} \times \frac{\mathsf{P}[S \setminus \{i, b_1, b_2, b_3\}]}{\mathsf{P}[S \setminus \{i, b_1, b_2\}]} \times \cdots \frac{\mathsf{P}[S \setminus \{i, b_1, b_2, b_3, \ldots, b_n\}]}{\mathsf{P}[S \setminus \{i, b_1, b_2, \ldots, b_{n-1}\}]}$$

Applying induction hypothesis to each of the terms in the RHS,

$$\frac{\mathsf{P}[S \setminus \Gamma^+(i)]}{\mathsf{P}[S \setminus \{i\}]} \leq \prod_{t \in \Gamma(i) \cap S} \frac{1}{(1 - x_j)}$$

Thus,

$$\frac{\mathsf{P}(S)}{\mathsf{P}[S \setminus \{i\}]} \geq 1 - x_i$$

and hence the proof. ∎

Application : Lower bounds for Ramsey Number: The application to Ramsey number is for the number $R(3, \ell)$. By the recurrence relation, it can be shown that $R(3, \ell) \leq \frac{\ell(\ell+1)}{2}$. We will obtain a lower bound for $R(3, \ell)$. The result is as follows.

THEOREM 4.4.9. *For $\ell \geq 3$, there is a constant c such that:*

$$R(3, \ell) \geq \frac{c\ell^2}{\log^2 \ell}$$

Proof. Since we are provinga lower bound, we should show that for n taken as the RHS of the theorem, there exists a 2-coloring that does not have a red triangle and a blue t-clique.

Let n be the number of vertices, which we will fix later. Consider a complete graph on n vertices and a random coloring of its edges with the colors red and blue: We color each edge red with probability p and blue with probability $1-p$; independently for all edges.[12]. We will want to prove that the probability that there is no red triangle and no blue K_ℓ is greater than 0.

Designing the Events: As planned, we will come up with bad events whose absence will imply the structure that we want. Indeed, bad events are :

- for $T \subseteq [n]$, $|T| = 3$, A_T is the event where vertices in T are colored with red color.
- for $S \subseteq [n]$, $|S| = \ell$, B_S is the event where vertices in S are colored with blue color.

We can also estimate the probabilities of these events. Indeed, $\Pr[A_T] \leq p^3$ and $\Pr[B_S] \leq (1-p)^{\binom{\ell}{2}}$. Also, $\overline{A_T}$ for all $T \subseteq [n]$ of size 3 and $\overline{B_S}$ for all $S \subseteq [n]$ of size ℓ implies that there is a 2-coloring which leaves no red triangle and blue K_ℓ as monochromatic. The number of vertices in the event graph is $\binom{n}{3} + \binom{n}{\ell}$.

Dependency graph: We now talk about the dependency graph. The above events are vertices. We add an edge between two events if their corresponding index sets intersect in more than one vertex (to get a common edge). That is an $(A_T, A_{T'}) \in E$ if and only if $|T \cap T'| > 1$ - and similarly for $(B_S, B_{S'}) \in E$ and for the cross edges (A_T, B_S). Notice that this satisfies the dependency relation as well. If there is no common edge between the index sets, then the corresponding events are independent too.

We estimate the degree of each vertex in the graph. Consider a vertex of type A_T where $T = \{u, v, w\}$. Each vertex can have the A-type neighbors and B-type neighbors. We will count the neighbours of A-type first. For each edge e in the triangle T, there can only be one more $A_{T'}$ such that $T \cap T'$ is exactly those two vertices. Hence the number A-type neighbors that A_T vertex can have, is at most $3n$.

We will count the B-type neighbors rather loosely. It can at most be the number of B-type vertices itself, which is $\binom{n}{\ell}$. Thus,

$$deg(A_T) \leq 3n + \binom{n}{\ell}$$

By similar consideration, for any $S \subseteq [n]$ with $|S| = \ell$.

$$deg(B_S) \leq n\binom{\ell}{2} + \binom{n}{\ell}$$

Satisfying the LLL premise: We now need to choose x_i's in LLL. Let us change to a convenient notation. We will call the x_is corresponding to A-type vertices of the dependency graph as x (we will choose all of them to be equal) and B-type vertices as the ys (we will choose all of them to be equal). Thus we have the following task in hand:

[12] Value of n and p will be spelt out later as expressions in terms of ℓ.

Task: We need to design n, p and $x, y \in (0,1)$ such that the inequalities (using the fact that $\Pr(A_T) = p^3$ and $\Pr[B_S] = (1-p)^{\binom{\ell}{2}}$:

$$p^3 \leq x(1-x)^{3n}(1-y)^{\binom{n}{\ell}} \tag{4.13}$$

$$(1-p)^{\binom{\ell}{2}} \leq y(1-x)^{n\binom{\ell}{2}}(1-y)^{\binom{n}{\ell}} \tag{4.14}$$

Staring at the expression $(1-y)^{\binom{n}{\ell}}$ hints that we should choose $y = \frac{1}{\binom{n}{\ell}}$ so that this expression can be replaced by roughly $\frac{1}{e}$.

Now since x and y are in $(0,1)$, Equation 4.14 gives us two inequalities.

$$p^3 \leq x \text{ and } p^3 \leq (1-x)^{\binom{n}{\ell}}$$

Using the fact that $(1-t)$ is roughly e^{-t}, the second equation gives, $p \geq xn \geq p^3$. Thus $p \geq \frac{1}{\sqrt[3]{n}}$ And Equation 4.14 gives us two inequalities.

$$(1-p)^{\binom{\ell}{2}} \leq (1-x)n\binom{\ell}{2} \text{ and } (1-p)^{\binom{\ell}{2}} \leq y = \frac{1}{\binom{n}{\ell}} = e^{-\ell \log n}$$

Applying $(1-t)$ is roughly e^{-t} again, $p\binom{\ell}{2} \geq \ell \log n$. Using $p\ell^2 \geq p\binom{\ell}{2}$ this gives, $\ell \geq \frac{\log n}{p} \geq \sqrt{n}\log n$. Motivated by these considerations, now we directly give the values for the variables involved.

$$n = \frac{\ell^2}{40\log^2 \ell} \text{ and } p = \frac{1}{3\sqrt{n}}$$

$$x = \frac{1}{9n^{3/2}} \text{ and } y = \frac{1}{\binom{n}{\ell}}$$

The constants are chosen to offset the error terms in the approximations considered. We leave it as an exercise to check that the inequalities are indeed satisfied.

Hence we have argued that for $n = \frac{\ell^2}{40\log^2 \ell}$ and $p = \frac{1}{3\sqrt{n}}$, we have a choice of the parameters of x and y that satisfies the constraints of the LLL. Hence,

$$\Pr\left[\left(\bigcap_{T:|T|=3} \overline{A_T}\right) \bigcap \left(\bigcap_{S:|S|=\ell} \overline{B_S}\right)\right] > 0$$

Hence there is a 2-coloring of K_n (for the above n) which leaves no red triangle and no blue K_ℓ in the graph. Hence the lower bound on the Ramsey number follows. □

REMARK 4.4.10. Although it may feel like the lower bound argument has a lot of slackness due to the choice of the parameters, it is independently known that $R(3,\ell) \leq \frac{c\ell^2}{\log^2 \ell}$.

Pairwise and t-wise Independence

Lecturer : Jayalal Sarma

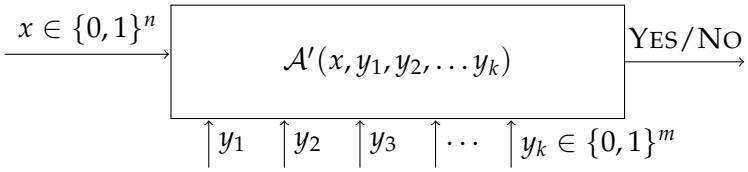
WEEK

5

As mentioned in the previous lecture, we do error reduction using limited independence now. We will show that if we use *pairwise independent* random variables, then it can still achieve error bounds of the form $\frac{1}{k}$ with only $O(\log n)$ additional random bits.

5.1 Pair-wise Independence

Although not related to expanders, we use this context to introduce one of the basic tools of pseudorandomness - *pairwise independent distributions*. We first recall the basic set up about amplification of success probability by repeating the algorithm.



In the trivial method, we supplied fully independent random strings $y_i \in \{0,1\}^m$ for each $i \in [k]$. Now we will allow some dependence among them. A set of random variables $Y_1, Y_2 \ldots Y_m$ is said to be *pairwise independent* if for all $i \neq j$, Y_i and Y_j are independent. That is,

$$\forall a, b : \Pr\left[(Y_i = a) \wedge (Y_j = b)\right] = \Pr\left[Y_i = a\right] \Pr\left[Y_j = b\right]$$

Constructing Pair-wise Independent Bits: We want k "dependent" random strings produced from fewer number of bits. We first do a simpler setting where instead of strings we ask for bits.

Suppose we want k bits of pairwise independent random bits. How many random bits do we need to spend? And from those pure random bits how do we produce the k bits? These questions are answered by an explicit construction now.

DEFINITION 5.1.1 (**Construction 1**). *We produce $k = 2^r - 1$ bits from r bits. The m Boolean random variables are indexed by $w \in \{0,1\}^r \setminus \{0^r\}$. The size of the space is 2^r. Fix $a \in \{0,1\}^r$, and for any $w \in \{0,1\}^r \setminus \{0^r\}$.*

$$Y_w = (\sum_i a_i w_i) \mod 2$$

We need to prove that the set of random variables are pairwise independent.

CLAIM 5.1.2. *The set of variables $\{Y_w \mid w \in \{0,1\}^r \setminus \{0^r\}\}$ are pairwise independent.*

Proof. We first prove the following two observations about the construction:

Subclaim 1 : For any fixed $w \in \{0,1\}^r \setminus \{0^r\}$, the random variable Y_w is uniformly distributed. Since $w \neq 0^r$, there must be i such that $w_i = 1$. Consider the non-empty subset of indices: $S = \{i \mid w_i = 1\}$. Hence $Y_w = 0$ if and only if $|\{i \mid a_i = 1 \wedge i \in S\}|$ is odd. Since the number of sequences $(a_i)_{i \in S}$ with even weight is exactly equal to the number of sequences $(a_i)_{i \in S}$ odd weight ones, the probability is exactly $\frac{1}{2}$.

Subclaim 2 : For any fixed $w, w \in \{0,1\}^r \setminus \{0^r\}$, such that $w \neq w'$, the random variable $Y_w \oplus Y_{w'}$ is uniformly distributed. This can be argued in a similar way to Subclaim 1. Only thing to note is that

$$Y_w \oplus Y_{w'} = \left(\sum_i a_i w_i \mod 2\right) \oplus \left(\sum_i a_i w'_i \mod 2\right) = \left(\sum_i a_i (w_i \oplus w'_i)\right) \mod 2 = Y_{w \oplus w'}$$

where $w \oplus w'$ is the bit-wise $\oplus$ of the bits in w and w'. Since $w \neq w'$, $w \oplus w' \in \{0,1\}^r$ and $w \oplus w' \neq 0$, and hence the same argument as in claim 1 applies.

Now we are ready to argue pairwise independence of Y_w and $Y_{w'}$. Let $w \neq w' \in \{0,1\}^r$. We need to show that for $\Pr[(Y_w = a) \wedge (Y_{w'} = b)] = \frac{1}{4}$. Let $\Pr[(Y_w = 0) \wedge (Y_{w'} = 1)] = p$. Since $w \neq w'$, there is an $i \in [r]$ such that $w_i \neq w'_i$. Hence,

$$\begin{aligned}
\text{Note that, } \frac{1}{2} = \Pr[(Y_w = 0)] &= \Pr[(Y_w = 0) \wedge (Y_{w'} = 0)] + \underbrace{\Pr[(Y_w = 0) \wedge (Y_{w'} = 1)]}_{p} \\
\Rightarrow \Pr[(Y_w = 0) \wedge (Y_{w'} = 0)] &= \frac{1}{2} - p \\
\text{Using this, } \frac{1}{2} = \Pr[(Y_{w'} = 0)] &= \underbrace{\Pr[(Y_{w'} = 0) \wedge (Y_w = 0)]}_{\frac{1}{2} - p} + \Pr[(Y_{w'} = 1) \wedge (Y_w = 1)] \\
\Rightarrow \Pr[(Y_w = 1) \wedge (Y_{w'} = 0)] &= p
\end{aligned}$$

Hence, $\Pr[Y_w \oplus Y_{w'} = 1] = 2p$. Because of subclaim 2, this gives $p = \frac{1}{4}$ and hence shows:

$$\Pr[(Y_w = a) \wedge (Y_{w'} = b)] = \frac{1}{4} = \Pr[(Y_w = a)] \Pr[(Y_{w'} = b)]$$

□

Constructing Pair-wise Independent Strings: While easy to analyze, the disadvantage of the inner product construction in the previous construction is that it outputs bits. However, for our purposes of amplification, we require pairwise independent strings $y_1, y_2, \ldots y_k \in \{0,1\}^m$. We now show a second construction which yields strings instead.

DEFINITION 5.1.3 (**Construction 2**). *Fix $q \geq k$. Choose $\alpha, \beta \in \mathbb{F}_q$ uniformly at random. Output $Y_w = \alpha w + \beta$ where $w \in \mathbb{F}_q$. This produces q bits from $2 \log q$ bits as seed. The size of the space is q^2.*

CLAIM 5.1.4. *The random variables $\{Y_w \mid w \in \mathbb{F}_q\}$ are pairwise independent.*

Proof. Let $w \neq w' \in \mathbb{F}_q$. Consider $a, b \in \mathbb{F}_q$. We need to show

$$\Pr\left[(Y_w = a) \wedge (Y_{w'} = b)\right] = \Pr\left[(Y_w = a)\right] \Pr\left[(Y_{w'} = b)\right] \tag{5.15}$$

Estimating RHS: We first observe that, for a fixed $w \in \mathbb{F}_q$, $\Pr\left[(Y_w = a)\right] = \frac{1}{q}$. To see this, note that for any $\alpha \in \mathbb{F}_q$, there is exactly one $\beta \in \mathbb{F}_q$ which satisfies $\alpha w + \beta = a$. Hence, there are q pairs (α, β) (among the q^2 possible pairs) which satisfies $\alpha w + \beta = a$. Hence RHS of Equation 5.15 is $\frac{1}{q^2}$.

Estimating LHS: The event $(Y_w = a) \wedge (Y_{w'} = b)]$ translates to the pair of equations: $\alpha w + \beta = a$ and $\alpha w' + \beta = a$. Since w, w' (and $w \neq w'$) and a, b are fixed, this is possible only for a unique pair of values $(\alpha, \beta) \in \mathbb{F}_q \times \mathbb{F}_q$. In other words the matrix equation:

$$\begin{pmatrix} w & 1 \\ w' & 1 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} a \\ b \end{pmatrix} \text{ and hence, } \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} w & 1 \\ w' & 1 \end{pmatrix}^{-1} \begin{pmatrix} a \\ b \end{pmatrix}$$

defines a bijection between the set of (α, β) pairs and (a, b) pairs. Hence, for every $(a, b) \in \mathbb{F}_q \times \mathbb{F}_q$, there is only one (α, β) (out of the q^2 choices) which will make $(Y_w = a) \wedge (Y_{w'} = b)$. Hence LHS of the Equation 5.15 is $\frac{1}{q^2}$.

□

Exercise 5.1.5 (See Problem Set 2 (Problem 5))**.** (a) For an $n \times m$ matrix A with Boolean entries and $b \in \{0,1\}^n$, define a function $h_{A,b} : \{0,1\}^m \to \{0,1\}^n$ by $h_{A,b}(x) = (Ax + b) \mod 2$ where the modulo 2 is applied component-wise. Show that

$$H_{m,n} = \{h_{A,b} \mid A \in \{0,1\}^{n \times m} \text{ and } b \in \{0,1\}^n\}$$

is a pairwise independent family of functions. Compare the number of random bits needed to generate a random function in $H_{m,n}$ to the construction that we did in class.

(b) A matrix A is a *Toeplitz matrix* if it is constant on diagonals, i.e., $A_{i+1,j+1} = A_{i,j}$ for all i, j. Show that even if we restrict the family $H_{m,n}$ in Part 1 to only include $h_{A,b}$ for Toeplitz matrices A, we still get a pairwise independent family. How many random bits are needed now?

5.2 Error Reduction using Pairwise Independence

We first describe the algorithm for amplification first. We choose $q = 2^m$: elements in $\mathbb{F}_q$ can be interpreted as strings of length m.

One immediate remark is that although it may look like we are wasting bits by generating more pairwise independent random bits than required, we cannot make use of them efficiently anyway since there are 2^m many of them.

We now prove the correctness lemma which indicates that if we need to reduce the error probability to $1/t$, then we need to repeat the algorithm for $k = O(t)$ times.

Algorithm 5.11 ($\mathcal{A}'$) : input $x \in \{0,1\}^n$

1: $count \leftarrow 0$.
2: Choose a pair $(\alpha, \beta) \in \mathbb{F}_q \times \mathbb{F}_q$ uniformly at random. ▷ (Uses $2m$ random bits).
3: Let $y_1, y_2, \ldots y_k$ be the first k elements of the pairwise independent space
$$\{\alpha w + \beta \mid w \in \mathbb{F}_q\}$$
4: **for** `each` $i \in [k]$ **do**
5: If $\mathcal{A}(x, y_i)$ accepts, if so increment *count*
6: **end for**
7: If $[count > \frac{k}{2}]$ then output YES else output NO.

LEMMA 5.2.1. *The probability of error for $\mathcal{A}'$ is bounded by:*

$$\Pr\left[\mathcal{A}' \text{ errs }\right] \leq \left(\frac{1}{\delta^2}\right)\frac{1}{k} \quad \text{where } \delta = \frac{1}{2} - \epsilon$$

Proof. As in the previous cases, for a given input $x \in \{0,1\}^n$, we define B to be the set of bad random strings - which makes the algorithm err on input x. We need to understand the probability that majority of $y_i \in B$. For this, let us define the indicator random variable.

$$Y_i = \begin{cases} 1 & \text{if } y_i \in B \\ 0 & \text{otherwise} \end{cases}$$

$\mathbb{E}[Y_i] = Pr[y_i \in B] = \epsilon$. We will define a translation of Y_i to make the expectation 0. Define $Z_i = Y_i - \epsilon$. Notice that $\mathbb{E}[Z_i] = 0$. In terms of these random variables, to bound the error for $\mathcal{A}'$, we want to analyze the probability that:

$$\Pr\left[\sum_{i=1}^{k} Y_i > \frac{k}{2}\right] = \Pr\left[\sum_{i=1}^{k} Z_i > k\left(\frac{1}{2} - \epsilon\right)\right] = \Pr\left[\sum_{i=1}^{k} Z_i > \delta k\right] = \Pr\left[Z > \delta k\right]$$

where $Z = \sum_{i=1}^{k} Z_i$. The random variable Z has expectation 0 and we want to estimate the probability that it is greater than δk. A natural situation is to apply Markov's inequality[13] which helps us estimate such a tail bound for positive random variables. Hence we plan to upper bound:

$$\Pr\left[Z > \delta k\right] \leq \Pr[Z^2 > \delta^2 k^2] \leq \frac{\mathbb{E}[Z^2]}{\delta^2 k^2}$$

Hence we need to estimate $\mathbb{E}[Z^2]$:

$$\begin{aligned} \mathbb{E}[Z^2] = \mathbb{E}\left[\left(\sum_{i=1}^{k} Z_i\right)^2\right] = \sum_{i=1}^{k} \mathbb{E}\left[Z_i^2\right] + 2\sum_{i<j} \mathbb{E}[Z_i Z_j] &\leq \sum_{i=1}^{k} \mathbb{E}\left[Z_i^2\right] + 2\sum_{i<j} \mathbb{E}[Z_i]\mathbb{E}[Z_j] \\ &\leq k \qquad \because \mathbb{E}[Z_i] = 0 \text{ and } \mathbb{E}[Z_i^2] \leq 1 \end{aligned}$$

[13] For a random variable X which takes only positive values in $\mathbb{R}$, $\Pr[X > a] \leq \frac{\mathbb{E}[X]}{a}$

Hence we have :

$$\Pr[Z > \delta k] \leq \frac{1}{\delta^2 k}$$

thus completing the proof. □

REMARK 5.2.2. As mentioned earlier, if we need $\frac{1}{t}$ as an upper bound on the error probability, we need to run the algorithm for $k = O(t)$ times.

Exercise 5.2.3. Let X be a random variable and $\mathbb{E}[X] = \mu$ be the expectation. Variance captures the expected square deviation from the mean μ. That is $\mathsf{Var}[X]$ is defined as $\mathbb{E}[(X-\mu)^2]$. It can be shown to be equal to $\mathbb{E}[X^2] - (\mathbb{E}[X])^2$ and that $\mathsf{Var}[aX] = a^2\mathsf{Var}[X]$. And more importantly, when the random variables $X_1, X_2, \ldots X_n$ are pairwise independent, then show that:

$$\mathsf{Var}[X_1 + X_2 + \ldots + X_n] = \mathsf{Var}[X_1] + \mathsf{Var}[X_2] + \ldots + \mathsf{Var}[X_n]$$

(Hint: Use the technique used in the proof of Lemma 5.2.1)

Provide an example that shows that the variance of the sum of two random variables is not necessarily equal to the sum of their variances, when the random variables are not independent. Indeed, one dramatic example when $n = 2$ is to take $X_2 = -X_1$. Find less dramatic ones !.

5.3 Derandomization using Pairwise Independent Bits

We now show an application of pairwise independent bits that we generated (instead of strings) in earlier section. This also will demonstrte algorithmic specific derandomization that we can do, at times, for certain problems.

Recall the MAXCUT problem. A cut in a graph is a partition of vertex set V into two $(S, \overline{S})$ and the size of a cut is the number of edges that go across the cut $|E(S, \overline{S})|$. The MAXCUT problem is as follows : *Given a graph $G(V, E)$ output the maximum cut in the graph.*

We will first give a simple randomized algorithm for the problem. Note that this is not a decision problem and hence the measure of how good the algorithm is based on the expected size of the cut that the algorithm outputs.

Algorithm 5.12 (*MaxCutAlgo*) : input $G(V, E)$, $|V| = n$

1: $S \leftarrow \phi$.
2: **for** each $w \in [V]$ **do**
3: Choose $b_w \in \{0, 1\}$ uniformly at random.
4: If $(b_w = 1)$ then add w to S.
5: **end for**
6: Output the cut $(S, \overline{S})$.

We quickly analyse the algorithm. For each edge $e = (u, v) \in E$ define the random variable X_e which will be 1 if the above algorithm makes the edge contribute to the cut and 0 otherwise. For a given edge the expected value of this random variable is :

$$\mathbb{E}[X_e] = \Pr[X_e = 1] = \Pr[(u \in S) \wedge (v \in \overline{S})] + \Pr[(u \in \overline{S}) \wedge (v \in S)] = \frac{1}{4} + \frac{1}{4} = \frac{1}{2} \tag{5.16}$$

The expected size of the cut then is obtained by estimating the expectation of $X = \sum_{e \in E} X_e$:

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

where OPT is the size of the maximum cut in the graph and it can be at most $|E|$. Thus the above algorithm gives a $\frac{1}{2}$ approximation for the MAXCUT problem on expectation.

The algorithm uses $O(n)$ random bits where $n = |V|$. We claim that the analysis of the algorithm still guarantees the expected size of the cut to be $\frac{1}{2}|OPT|$ even if we do not use pure randombits, but instead supply the pairwise indepednent randbom bits we generate (from definition 5.1.1). We choose $n = 2^r - 1$ which uses $r = O(\log n)$ seed in order to choose from the pairwise independent space that we constructed. Let the bits that we provided by $b_1, b_2, \ldots b_n$ produced from construction 5.1.1.
Equation 5.16 can be rederived as:

$$\begin{aligned}\mathbb{E}[X_e] = \Pr[X_e = 1] &= \Pr[(b_u = 1) \wedge (b_v = 0)] + \Pr[(b_u = 0) \wedge (b_v = 1)] \\ &= \Pr[b_u = 1]\Pr[b_v = 0] + \Pr[b_u = 0]\Pr[b_v = 1] = \frac{1}{2}.\frac{1}{2} + \frac{1}{2}.\frac{1}{2} = \frac{1}{2}\end{aligned}$$

were we use the pairwise independence in the third equality (which we wrote earlier because of full independence.

Hence, we have an algorithm which uses $O(\log n)$ random bits (call the string y') and provides a $\frac{1}{2}$-approximation for the maximum cut on expectation. Hence there must be a random string setting for y' which produces the $\frac{1}{2}$-approximation such that $\left|E(S, \overline{S})\right| \geq \frac{1}{2}|OPT|$. Hence, this leads to a polynomial time deterministic algorithm, which runs over all the possible assignments for the string y' (only $\mathsf{poly}(n)$ in number) and runs the algorithm using each setting of y' as random string, and outputs the maximum sized cut produced. By the above argument, this algorithm is guranteed to produce a $\frac{1}{2}$-approximation for the maximum cut in the graph.

Exercise 5.3.1. Recall the discussion on Ramsey graph's where we applied probablistic method to show bounds on Ramsey number $R(k, k)$. Let us define a graph to be a *k-Ramsey graph* if it has no clique or independent set of size at least k. We had shown by the probabilistic method that there exists a graph on n vertices that $O(\log n)$-Ramsey. An interesting open problem is to give an *explicit* construction of a $O(\log n)$-Ramsey graph. The best known construction runs in time $O(2^{2^{\log^{\epsilon} n}})$ for some small $\epsilon = 0.99$. In this problem you are asked give a construction for $O(\log n)$ Ramsey graph that runs in time $O(n^{\log^2 n})$. [Hint : Check if full independence of random choices is necessary in our proof by probabilistic method. Show that the proof actually gives a set of $O(n^{\log^2 n})$ graphs, one of which is guaranteed to be $O(\log n)$-Ramsey]

Curiosity 5.3.2. The best approximation ration for maximum cut problem is not $\frac{1}{2}$. It is 0.878, and it comes from the semidefinite programming (SDP) relaxation for maxcut problem LP. Define $n + m$ variables x_u for each $u \in V$ and $e_{uv} \in E$. These variables are supposed to represent the information $e_{uv} = 1$ if and only if (u, v) is in the cut, and $x_u = 1$ if and only if $u \in S$. The LP objective function is to maximise $\sum_{(u,v) \in E} e_{uv}$ subject to the constraints (1) $\forall u, v \in E$, $e_{uv} \leq x_u + x_v$ and $e_{uv} \leq 2 - (x_u + x_v)$, (2) all of them are Boolean variables. The idea due to [Goemans and Williamson, 1995]

is to relax this condition (2) by using vector valued variables (rather than Boolean). Not only that this relaxation can be solved efficiently, but [Goemans and Williamson, 1995] gave a method of rounding the variables to Boolean values and proving that the approximation ratio is atleast $0.878\ldots$.

Complementing this, an optimal inapproximability result was proven for MAXCUT by [Khot et al., 2007], based on a conjectured hardness for the approximation problem known as the label cover problem (this is also called the *unique games conjecture* (UGC) - see [Khot, 2010]).

Curiosity 5.3.3.

2: Jayalal says: Todo - Write about Luby's algorithm

5.4 t-wise Independence Spaces

The notion of pairwise independence that we learned in the previous section can be generalized further. We say that a set of random variables $Y_1, Y_2, \ldots Y_k$ to be t-wise independent if any subset of t random variables among them are independent. In other words, for any $I \subseteq [k]$ such that $|I| \leq t$, and any set of distinct elements $a_1, a_2, \ldots a_k$

$$\Pr\left[\bigwedge_{i\in I}(Y_i = a_i)\right] = \prod_{i\in I}\left(\Pr\left[Y_i = a_i\right]\right) \tag{5.17}$$

DEFINITION 5.4.1 (**Constructing t-wise Independent Strings**). *Fix $q \geq k$. Choose $c_0, c_1, \ldots c_{t-1} \in \mathbb{F}_q$ uniformly at random. Output $Y_w = c_0 + c_1 w + c_2 w^2 + \ldots c_{t-1} w^{t-1}$ where $w \in \mathbb{F}_q$. This produces q bit string from $O(t \log q)$ bit seed. The size of the space is q^t.*

CLAIM 5.4.2. *The random variables $\{Y_w \mid w \in \mathbb{F}_q\}$ are t-wise independent.*

Proof. For any $I \subseteq [k]$ such that $|I| \leq t$, and any set of elements distinct $a_1, a_2, \ldots a_k \in \mathbb{F}_q$, we need to show, Equation 5.17.

Estimating RHS: We first observe that, for a fixed $w \in \mathbb{F}_q$, $\Pr\left[(Y_w = a)\right] = \frac{1}{q}$. To see this, note that for any $c_1, c_2, \ldots c_k \in \mathbb{F}_q$, there is exactly one $c_0 \in \mathbb{F}_q$ which satisfies $c_0 + c_1 w + c_2 w^2 + \ldots c_{t-1} w^{t-1} = a$. Hence, there are q different t-tuples $(c_0, c_1, \ldots c_{t-1})$ (among the q^t possible pairs) which satisfies $c_0 + c_1 w + c_2 w^2 + \ldots c_{t-1} w^{t-1} = a$. Hence RHS of Equation 5.17 is $1/q^t$.

Estimating LHS: Let the index set be $I = \{w_1, w_2, \ldots w_t\}$. The event $(Y_{w_1} = a_1) \wedge (Y_{w_2} = a_2) \wedge (Y_{w_3} = a_3) \ldots \wedge (Y_{w_t} = a_t)]$ translates to the t equations:

$$\begin{aligned}
c_0 + c_1 w_1 + c_2 w_1^2 + \ldots c_{t-1} w^{t-1} &= a_1 \\
c_0 + c_1 w_2 + c_2 w_2^2 + \ldots c_{t-1} w^{t-1} &= a_2 \\
&\vdots \quad \vdots \\
c_0 + c_1 w_t + c_2 w_t^2 + \ldots c_{t-1} w^{t-1} &= a_t
\end{aligned}$$

Since I is fixed, w_is are fixed, and given tuples $(a_1, a_2, \ldots a_t)$, there is exactly one tuple

$(c_0, c_1, \ldots c_{t-1})$ which satisfies the above equation. This can also be written as the matrix equation:

$$\begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_{t-1} \end{pmatrix} = \begin{pmatrix} 1 & w_1 & w_1^2 & \ldots & w_1^{t-1} \\ 1 & w_2 & w_2^2 & \ldots & w_2^{t-1} \\ \vdots & \vdots & \vdots & \vdots & \\ 1 & w_t & w_t^2 & \ldots & w_t^{t-1} \end{pmatrix}^{-1} \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_t \end{pmatrix}$$

The matrix is invertible since it is in the Vandermonde form with distinct rows. Hence, for every $(a_1, a_2 \ldots a_t) \in \mathbb{F}_q^t$, there is only one $(c_0, c_1, \ldots c_{t-1})$ (out of the q^t choices) which will make $Y_{w_i} = a_i$ for all $i \in [t]$. Hence LHS of the Equation 5.15 is $\frac{1}{q^t}$.

□

5.5 Error Reduction using t-wise Independence

Again we choose $q = 2^m$: elements in $\mathbb{F}_q$ can be interpreted as strings of length m.

Algorithm 5.13 ($\mathcal{A}'$) : input $x \in \{0,1\}^n$

1: $count \leftarrow 0$.
2: Choose $(c_0, c_1, \ldots c_{t-1}) \in \mathbb{F}_q^t$ uniformly at random. ▷ (Uses tm random bits).
3: Let $y_1, y_2, \ldots y_k$ be the first k elements of the pairwise independent space
$$\{c_0 + c_1 w + c_2 w^2 + \ldots c_{t-1} w^{t-1} \mid w \in \mathbb{F}_q\}$$
4: **for** `each` $i \in [k]$ **do**
5: If $\mathcal{A}(x, y_i)$ accepts, if so increment $count$
6: **end for**
7: If $[count > \frac{k}{2}]$ then output YES else output NO.

We now prove the correctness lemma which indicates that if we need to reduce the error probability to $1/t$, then we need to repeat the algorithm for $k = O(t)$ times.

LEMMA 5.5.1. *The probability of error for $\mathcal{A}'$ is bounded by:*

$$\Pr\left[\mathcal{A}' \text{ errs}\right] \leq \left(\frac{1}{\delta^t}\right) \frac{1}{k^{t/2}} \quad \text{where } \delta = \frac{1}{2} - \epsilon$$

Proof. The proof and the notations are same as Proof of Lemma 5.2.1 in the case of pairwise independence. We present only the different parts here. Y_i is the indicator random variable, as defined, and $Z_i = Y_i - \epsilon$ and $\mathbb{E}[Z_i] = 0$. In terms of these random variables, to bound the error for $\mathcal{A}'$, we want to analyze the probability that:

$$\Pr\left[\sum_{i=1}^{k} Y_i > \frac{k}{2}\right] = \Pr\left[\sum_{i=1}^{k} Z_i > k\left(\frac{1}{2} - \epsilon\right)\right] = \Pr\left[\sum_{i=1}^{k} Z_i > \delta k\right] = \Pr\left[Z > \delta k\right]$$

where $Z = \sum_{i=1}^{k} Z_i$. Hence we plan to upper bound (assume that t is even:

$$\Pr[Z > \delta k] \leq \Pr[Z^t > \delta^t k^t] \leq \frac{\mathbb{E}[Z^t]}{\delta^t k^t}$$

Hence we need to estimate $\mathbb{E}[Z^t]$:

$$\begin{aligned}\mathbb{E}[Z^t] = \mathbb{E}\left[\left(\sum_{i=1}^{k} Z_i\right)^t\right] &= \sum_{i=1}^{k} \mathbb{E}\left[Z_i^t\right] + (\text{terms with at least one odd degree } \mathbb{E} \text{ term}) \\ &\quad + (\text{terms with all even degree } \mathbb{E} \text{ terms})\end{aligned}$$

Thus, the only terms that survive are those where every term appears an even number of times. For example, when $t = 4$,

$$\begin{aligned}\mathbb{E}[Z^4] &= \sum_{i=1}^{k} \mathbb{E}\left[Z_i^4\right] + \binom{4}{2} \sum_{0 \leq i < j \leq k} \mathbb{E}[Z_i^2]\mathbb{E}[Z_j^2] \\ &\leq k + \binom{4}{2}\binom{k}{2} \leq 4k^2\end{aligned}$$

$$\begin{aligned}\text{For a general } t \leq k, \mathbb{E}[Z^t] &\leq O(k^{t/2}) \\ \text{Hence, } \Pr[Z > \delta k] &\leq O\left(\frac{1}{k^{t/2}}\right)\end{aligned}$$

Thus, if we take 2ℓ-independence, the error reduces to $O(1/k^\ell)$. If we are targeting to get to an error bound of γ then the number of repetitions must be $\sqrt[\ell]{1/\gamma}$. This is indeed better than the number of repetitions required if we were using pairwise indepedence space. □

We conclude this part by providing a table of the various amplifcation methods that we have seen so far. Note that ϵ is less than half and δ denotes how close it is to $\frac{1}{2}$. That is, $\delta = \frac{1}{2} - \epsilon$.

Approach	Error bound	Random bits used
Trivial	$\frac{1}{2}(1 - 4\epsilon^2)^{k/2}$	km
Pairwise Independence	$\frac{1}{k\delta^2}$	$2m$
t-wise Independence	$\frac{1}{k^{t/2}\delta^t}$	tm

In the trivial method, the dependence on k is on the exponent, and hence we will need to do only logarithmic (in the target error) repetitions in order to achieve a given error bound. But in the pairwise independence, the dependence on k is linear and hence we will have to do linear (in the target error bound) number of repetitions. And t-wise independence strikes the middle of these two.

Expander Graphs : Existence and Algebraic Constructions

Lecturer : Jayalal Sarma

WEEK

6

We now use the tools that we have developed in order to construct show existance and then explict constructions of expanders. We recall the definition of expanders from Section 3.3 but now we make the paramters more precise.

DEFINITION 6.0.2. *(**Expander Graphs)** A graph $G(V, E)$ is said to be a (n, d, α, β)-expander if, $|V| = n$, it is d-regular, and every $S \subseteq V$ such that $|S| \leq \alpha|V|$, the number of new neighbors $|N(S) \setminus S| \geq \beta d|S|$.*

Note : We are changing the notation slightly from what we followed in the lecture to ensure that both α and β are in $(0, 1)$.
A weaker setting is to consider bipartite graphs where we insist the above only for subset of vertices from one side of the bipartite graph, so that $|N(S) \setminus S| = |N(S)|$ since there is no edge from a set to itself in this case.

DEFINITION 6.0.3. *(**Bipartite Expander Graphs)** A graph $G(U \cup V, E)$ is said to be a (n, d, α, β)-expander if, $|U| = n$, it is d-regular and every $S \subseteq U$ such that $|S| \leq \alpha|U|$, the number of new neighbors $|N(S)| \geq \beta d|S|$.*

6.1 Existence of Left regular Bipartite expanders

What kind of parameters should we expect for expanders and what do we need. Looking at the possible application in the case of randomness efficient amplification, we would like the degree of the graph to be much much smaller than the number of vertices (ideally a constant). It is natural to keep the graph regular because then the mathematical constraints are easier to handle (as we will see). We can also allow the graph to have multiple edges between two vertices since we do not require the neighbors to be distinct.

Ideally, we would want β to be as close to 1 as possible. In fact, for many of the applications, it is enough to have $\beta > \frac{1}{2}$. We will show the existance of a $(n, d, \alpha, \frac{d-2}{d})$ expander which will be good as long as $d \geq 5$. We show the following theorem (which not only shows existence but also implies abundance).

THEOREM 6.1.1. *For every constant d, there is an $\alpha > 0$ such that for all large enough n: a uniformly chosen random bipartite graph $G(U \cup V, E)$ which is left d-regular and $|U| = |V| = n$, is an $(n, d, \alpha, \frac{d-2}{d})$ with probability at least $\frac{1}{2}$.*

Proof. We spell out the details of the experiment first. Each vertex $u \in U$ is assigned d neighbors from the right side (with replacement) chosen uniformly at random, each indepedently.

We will compute probability that the outcome of the experiment is not an expander and show that this probability is less than $\frac{1}{2}$.
The graph is not an expander with the required parameters if there exists a set $S \subseteq U$, $|S| \leq \alpha n$ such that $|N(S)| < (d-2)|S|$. We will use union bound over all such S.

$$\Pr\left[\begin{array}{l}\exists S \subseteq U, \text{ with } |S| \leq \alpha n \\ \text{s.t. } |N(S)| < (d-2)|S|\end{array}\right] \leq \sum_{\substack{S \subseteq U \\ |S| \leq \alpha n}} Pr[|N(S)| < (d-2)|S|] \leq \sum_{k=1}^{\alpha n} \binom{n}{k} p_k$$

where p_k is the upper bound we hope to derive for $Pr[|N(S)| < (d-2)|S|]$ for any $S \subseteq U$ with $|S| = k$.

Now, we just need to estimate p_k. First, we observe that the only way the number of neighbors of a vertex $u \in U$ can go below d is when the random choice ends up choosing the same vertex $v \in V$ more than once in the experiment for u. To use this, let $M = \{\{v_1, v_2, \ldots v_{kd}\}\}$ be the multiset of kd outcomes when d neighbors are chosen uniformly at random for each of the vertices in S. If $|N(S)| < (d-2)k$, then it must be because there exists $2k$ repetitions in this sequence. The repetitions can be any of the $2k$ sub-multi-subsets out of these kd neighbors. We apply union bound over all the subsets. Thus,

$$\begin{aligned} p_k = \Pr\left[\begin{array}{l}\exists T \subseteq M, \text{ with } |T| = 2k \\ \text{s.t. all elements in } T \text{ are repeats}\end{array}\right] &= \sum_{\substack{T \subseteq M \\ |T|=2k}} \Pr\left[\begin{array}{l}\text{all elements in } T \\ \text{are repeats}\end{array}\right] \\ &\leq \sum_{\substack{T \subseteq M \\ |T|=2k}} \left(\frac{kd}{k}\right)^{2k} \leq \binom{kd}{2k}\left(\frac{kd}{n}\right)^{2k} \end{aligned}$$

The last inequality is by noting that the probability of any particular element repeating is at most $\frac{kd}{n}$ (the worst case is for the last element in T chosen). Hence,

$$\begin{aligned} \Pr\left[\begin{array}{l}\exists S \subseteq U, \text{ with } |S| \leq \alpha n \\ \text{s.t. } |N(S)| < (d-2)|S|\end{array}\right] &\leq \sum_{k=1}^{\alpha n} \binom{n}{k}\binom{kd}{2k}\left(\frac{kd}{n}\right)^{2k} \\ &\leq \sum_{k=1}^{\alpha n} \left(\frac{ne}{k}\right)^k \left(\frac{kde}{2k}\right)^{2k}\left(\frac{kd}{n}\right)^{2k} \qquad \text{using } \binom{n}{k} \leq \left(\frac{ne}{k}\right)^k \\ &\leq \sum_{k=1}^{\alpha n} \left(\frac{e^3 d^4 k}{4n}\right)^k \leq \sum_{k=1}^{\alpha n} 4^{-k} \qquad \text{choose } \alpha = \frac{1}{e^3 d^4} \\ &< \frac{1}{2} \end{aligned}$$

Hence the proof. ∎

6.2 Variants of Expanders

The discussion in the beginning of the previous section motivates the definition of variants of expander definition. What we defined in the previous section is called the boundary expansion where we insisted that $|N(S) \setminus S|$ must be large. We define two variants of the same.

DEFINITION 6.2.1 (**Vertex Exapansion**). *A graph $G(V, E)$ is said to be a (n, d, α, β)-expander if, $|V| = n$, it is d-regular, and every $S \subseteq V$ such that $|S| \leq \alpha|V|$, the number of new neighbors*

$$|N(S)| \geq \beta d|S|$$

Notice that when we consider bipartite graphs $N(S) \cap S \neq \phi$, but still the above definition is stronger since it asks for all $S \subseteq V$ and not the subsets only in one side.

DEFINITION 6.2.2 (**Edge Exapansion**). *A graph $G(V, E)$ is said to be a (n, d, α, β)-expander if, $|V| = n$, it is d-regular, and every $S \subseteq V$ such that $|S| \leq \alpha|V|$, the number of new neighbors*

$$|E(S, \overline{S})| \geq \beta d|S| \tag{6.18}$$

Viewing it from the graphs side, we can define the edge expansion ratio of a d-regular graph G as,

$$h(G) = \min_{\{S \subseteq [n] : |S| \leq \frac{n}{2}\}} \frac{|E(S, \overline{S})|}{d|S|}$$

In other words, for a given graph G, what is the smallest β for which it satisfies Equation 6.18. The answer is the edge expansion ratio $h(G)$.

Exercise 6.2.3. Let $n, d \in \mathbb{N}$, $\alpha, \beta > 0$. Every (n, d, α, β)-boundary-expander is also a $(n, d, \alpha, \frac{\beta}{d})$-edge-expander. Conversely, every (n, d, α, β)-edge-expander is also a (n, d, α, β)-boundary-expander.

Exercise 6.2.4 (See Problem Set 3 (Problem 1))**.** Let $G = (V, E)$ be a graph. For every subset S of its vertices V we say, that vertex $v \in V \setminus S$ is a neighbor of S if $E(S, v) \geq 1$, an odd neighbor of S if $E(S, v)$ is odd, a unique neighbor of S if $E(S, v) = 1$. Further, we denote:

$$\begin{aligned} B(S) &= \{v \in V \setminus S \mid v \text{ is a neighbor of } S\} \\ B_{odd}(S) &= \{v \in V \setminus S \mid v \text{ is an odd neighbor of } S\} \\ B_{unique}(S) &= \{v \in V \setminus S \mid v \text{ is a unique neighbor of } S\} \end{aligned}$$

A graph G is said to be an (n, d, α, β)-odd-neibhor (resp. unique-neighbor) expandor if for every $S \subseteq V$, where $|S| \leq \alpha n$, the set B_{odd} (resp. B_{unique}) is of size at least $\beta d|S|$.

(a) Show that every (n, d, α, β) unique-neighbor expander is an (n, d, α, β) odd-neighbor expander.

(b) Show that every (n, d, α, β) odd-neighbor expander is also an (n, d, α, β) boundary expander.

(c) Show that every $(n, d, \alpha, \frac{1}{2} + \frac{\epsilon}{d})$-boundary expander is also a $(n, d, \alpha, \frac{2\epsilon}{d})$-unique-neighbor expander.

It is usually assumed that $\alpha = \frac{1}{2}$. In fact, if we have an edge expander with $\alpha = \frac{1}{2}$, then we have one for larger α as well. The following exercise will ascertain that.

Exercise 6.2.5 (See Problem Set 3 (Problem 2))**.** Let $n \in \mathbb{N}$ and $\frac{1}{2} < \alpha < \frac{n-1}{n}$, and $G(V, E)$ by an $(n, d, \frac{1}{2}, \beta)$ edge-expander, then G is also an $(n, d, \alpha, (1-\alpha)\beta)$ edge expander.

We showed in the last section that, bipartite expanders exist with good parameters. Now we will show edge-expanders exists using the expectation method. The following problem is designed for that.

Exercise 6.2.6 (See Problem Set 3 (Problem 3))**.** Through the following steps, show that there exists a family of $(n, d, \frac{1}{2}, \beta)$ edge-expander.

(a) Choose β later. Choose a random d-regular graph on n vertices. Let $S \subseteq V$, with $s = |S| \leq \frac{n}{2}$. Define the random variable $E_S = |E(S, V \setminus S)|$. Let $0 \leq k \leq \beta d|S|$. Prove that if $sd - k$ is an odd number, then, $\Pr[E_S = k] = 0$.

(b) If $sd - k$ is even, then show that

$$\Pr[E_S = k] \leq \left(\frac{3s}{2n}\right)^{ds/4} \qquad \text{Use } \binom{a}{b}\left(\frac{ae}{b}\right)^b \text{ and choose } \beta \text{ such that } \left(\frac{e}{\beta}\right)^{4\beta} \leq \frac{9}{8}$$

(*Hint: Suppose $sd - k$ is even, to calculate probability of k edges leaving the set - select which edges are those k, matching them with any of the $nd - sd$ possible endpoints outside of the set S. Every edge with one endpoint in the subset S has probability roughly s/n that the other endpoint is contained in S as well.*).

(c) Use the previous part to show that :

$$\Pr\left[E_S < \beta ds\right] \leq \left(\frac{5s}{3n}\right)^{ds/4} \qquad \text{Assume } d \geq 140 \text{ and approximate : } \frac{sd}{4} \leq \left(\frac{10}{9}\right)^{sd/4}$$

(d) Prove that the probability that there exists a set of size at most $\frac{n}{2}$, is < 1. Hence conclude the theorem.

6.3 Marguilis-Gabber-Galil Expander

Although we showed the existence arguments only for left-regular bipartite expanders, similar arguments exists for other kinds of objects in the above lists too. Now we turn into explicit constructions of families of expander graphs.

There are two major approaches to constructing expander graphs. One regime is purely algebraic, in which the expander graphs are defined over algebraic structures and connectivity is determined by algebraic properties. They are very easy to describe (more formally, they are very explicit). However, the arguments that they are indeed expanders is more complicated (in fact, even beyond the scope of this course, in terms of background required). Hence we will not be proving that they are expanders, but we will describe them.

The other regime of constructions is more combinatorial and are based on graph operations. It takes care to make them explicit, but the proof of expansion is somewhat more amenable for the background of this course.

Marguilis-Gabber-Galil Expander: First family was studied by [Margulis, 1973] where the proof of expansion was based on representation theory and did not provide any specific bound on the expansion ratio h. Later [Gabber and Galil, 1981] derived such a bound using harmonic analysis.

We now describe the family which is on n^2 vertices is $V = \mathbb{Z}_n \times \mathbb{Z}_n$. The neighbors of the vertex $(a, b) \in V$ are :

$$N(a,b) = \left\{ \begin{array}{l} (a+b, b) \\ (a-b, b) \\ (a, b+a) \\ (a, b-a) \\ (a, b+a+1) \\ (a, b-a+1) \\ (a+b+1, b) \\ (a-b+1, b) \end{array} \right\}$$

Expanders from Number Theory: These graphs can be constructed when the number of vertices expected is a prime. The family of 3-regular contains p-vertex graphs for every prime p. Here $V = \mathbb{Z}_p$, and for a vertex $a \in V$:

$$N(a) = \{a+1, a-1, a^{-1}\}$$

where all operations are in $\mathbb{Z}_p$.

Although we will not describe the proof of expansion of the above graphs, they all go through a special connection between edge expansion and the spectrum of a graph. We quickly introduce this in the next section.

6.4 Spectral Expansion

We consider simple graphs for simplicity. For a d-regular undirected graph, define the normalized adjacency matrix as follows:

$$A_{uv} = \left\{ \begin{array}{ll} \frac{1}{d} & (u,v) \in E \\ 0 & \text{otherwise} \end{array} \right.$$

Since G is undrected, the matrix is symmetric and hence the eigen values of the matrix are all real numbers[14]. Since the graph is d-regular, the matrix is also doubly stochastic. Hence the all 1 vector (and any scalar multiple of it) is an eigen vector and 1 is the eigen value corresponding to it. The largest eigen value of a doubly stochastic matrix is 1 (Prove this as an exercise !). Notice that there are at most n eigen values and some of them could be with higher multiplicity.

[14]We reviewed the basic related definitions about eigen values in the lecture, but we are not writing them here to avoid digression.

All eigen values of the matrix are in the interval $[-1, 1]$. Folding this range, by taking absolute values, let the absolute values of the eigen values of the normalized adjacency matrix[15] $1 = \lambda_1 \geq \lambda_2 \geq \lambda_3 \ldots \lambda_k \geq 0$ where $k \leq n$. This sequence is called the *spectrum of the graph*. Spectrum of a graph contains surprising information about the combinatorial structure of the graph. Spectral graph theory studies the relationship between spectrum and combinatorial properties of the graph (for example, connectedness, bipartiteness, number of connected components).

The gap between the first two eigen values is called the **spectral gap** of a graph. And a d-regular graph is called a spectral expander if the second largest eigen value $\lambda_2(G)$ is below 1 by a constant gap. More formally:

DEFINITION 6.4.1 (**Spectral Expanders**). *A d-regular graph is said to be (n, d, λ) spectral expander if $\lambda_2(G) \leq \lambda$. Equivalently, the spectral gap of the graph is at least $1 - \lambda$.*

There are two questions that we will address now. First of all, why study spectral expansion? We show that spectral expanders in fact have edge expansion too. Thus to show that a graph is an edge expander, it suffices to show that it is an spectral expander. This is the approach taken for proving that the algebraic constructions the we described in the previous section are expanders. However, the way to achieve that is still through harmonic anlaysis and number theory etc.

The second question is to do the curious statement that spectral expanders are in fact edge expanders too. We try to make this precise through the following statement.

THEOREM 6.4.2 (**Spectral Expansion Implies Edge Expansion**). *If G is an (n, d, λ) spectral expander, then it is also a $(n, d, \frac{1}{2}, \frac{1-\lambda}{2})$ edge expander.*

In fact, the converse of the above theorem is also true. We show the formal proof of the above statement in the next lecture (in fact, a much stronger form too) with the converse.

In the rest of this lecture, we make an informal handwavy attempt to justify why one should expect such a connection between spectral expansion and edge expansion. The argument is through the idea of random walks on graphs. A random walk on a graph G is a random experiment performed on the G as follows. Starting from a vertex v and choosing the next neighbor among the neighbors of the current vertex. The walk produces a sequence of vertices. The relevant question is - fix an integer $\ell \in \mathbb{N}$ - *what does the distribution of the ℓ^{th} vertex in the walk look like?* Of course, the answer depends on the graph. Then the question is what parameter of the graph governs the distribution of the vertex after ℓ steps of the walk? Are there graphs for which it will be the uniform distribution or even close to uniform distribution.

If the distribution gets closer and closer to uniform - then it is termed as rapid mixing of the random walk. The rate at which it gets closer is a paramter. If there was no edge expansion, then we would have had random walks initiating from the set S (which did not have edge expansion) to not get distributed uniform easily. Hence we would expect random walks to not mix fast. In the contrapositive, this says that, intuitively, *if random walks in a graph mixes well, then there must be edge expansion in the graph*.

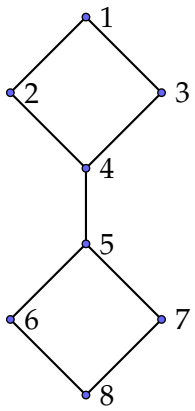
Now we need to address the question, why does spectral gap imply that random walks in the graph mix well? This has a very cute linear algebraic reason. To answer this at least informally, we

[15] We denote them as $\lambda_i(G)$.

take an example graph. To keep track of the distribution that evolves with the length of the walk, we denote by $p_i \in \mathbb{R}^n$ the probability distribution vector after the ith step in the random walk.

Suppose we are starting from the topmost vertex. Let X_i be the random variable denoting the vertex after the i^{th} step. Define the vector p_i, with entries as $p_i[j] = \Pr[X_i = j]$.

	1	2	3	4	5	6	7	8
p_0	1	0	0	0	0	0	0	0
p_1	0	0.5	0.5	0	0	0	0	0
p_2	0.5	0	0	0.5	0	0	0	0
p_3	0	0.42	0.42	0	0.17	0	0	0
$\vdots$			$\vdots$			$\vdots$		



$$p_{i+1}[j] = \Pr[X_{i+1} = j] = \sum_k \Pr[X_i = k] \Pr[\text{ } (k,j) \text{ edge is chosen }]$$

This implies that $p_{i+1} = Ap_i$.

Now let $v_1, v_2, \ldots v_n$ be an orthonormal basis of $\mathbb{R}^n$ where each v_i is an eigen vector of λ_i. By definition any vector can be written as a linear combinationof vectors in $\mathbb{R}^n$. Let $p_i = \alpha_1 v_1 + \alpha_2 v_2 + \cdots + \alpha_n v_n$.

$$\begin{aligned} p_{i+1} = Ap_i &= \alpha_1 Av_1 + \alpha_2 Av_2 + \cdots + \alpha_n Av_n \\ &= \alpha_1 v_1 + \alpha_2 \lambda_2 v_2 + \cdots + \alpha_n \lambda_n v_n \end{aligned}$$

Notice that the component corresponding to the eigen value 1, remains unchanged in the vector,and the other components gets multiplied by at least λ_2 and hence gets reduced. Thus the vector will go closer and closer to uniform distribution as the walk goes further and further. The rate of mixing (to uniform distribution) will depend on how small $\lambda_2(G)$ is. Thus, *if there is a constant spectral gap, then the random walk mixes fast.*

Together with the earlier statement - *if random walks in a graph mixes well, then there must be edge expansion in the graph*, this implies that we should expect a connection between second largest eigen value and edge expansion ratio. This is indeed, the topic of discussion for next lecture.

Curiosity 6.4.3 (**Gershgorin's Circle Theorem**)**.** This is a digression. We used the following statement (where we left the proof as an exercise). *The largest eigen value of a doubly stochastic matrix is* 1 For graphs without self-loops, this is a special case of a more general theorem called *Gershgorin circle theorem* may be used to bound the spectrum of a square matrix. It was first published by the Gershgorin in 1931. The statement is as follows:

Let A be an $n \times n$ matrix with entries from $\mathbb{C}$, with entries a_{ij}. For $i \in [n]$ let $R_i = \sum_{j \neq i} |a_{ij}|$ be the sum of the absolute values of the non-diagonal entries in the i-th row. Let $D(a_{ii}, R_i) \subseteq \mathbb{C}$ be a closed disc centered at a_{ii} with radius R_i. Such a disc is called the *Gershgorin disc*.

Gershgorin's Circle Theorem : *Every eigenvalue of A lies within at least one of the Gershgorin discs.*

Indeed, in our case, for every i, $R_i = 1$, $a_{ii} = 0$. This immediately implies the statement we wanted to conclude. For a diagonal matrix, the Gershgorin discs coincide with the spectrum.For a diagonal matrix, the Gershgorin discs coincide with the spectrum. Conversely, if the Gershgorin discs coincide with the spectrum, the matrix is diagonal.

6.5 Cheeger's Inequality

We described the vague reason why we might expect that graphs with a constant spectral gap should be expected to be good edge expanders as well. Now, we make this precise.

Recall from the previous lecture about the edge expansion ratio.

$$h(G) = \min_{\{S \subseteq [n] : |S| \leq \frac{n}{2}\}} \frac{|E(S, \overline{S})|}{d|S|}$$

Cheeger's inequality establishes a strong connection between the edge expansion ratio and the the second largest eigen value.

THEOREM 6.5.1 (**Cheeger's Inequality**). *For any undirected d-regular graph G:*

$$\frac{1 - \lambda_2(G)}{2} \leq h(G) \leq \sqrt{2(1 - \lambda_2(G))}$$

The idea of the proof is to use a new parameter called *conductance* of the graph which is closely related to edge expansion ration and then it use it to prove the bound. We define conductance first.

$$\Phi(G) = \min_{\{\phi \subseteq S \subseteq [n]\}} \frac{|E(S, \overline{S}|}{d|S| \left(|\overline{S}|/|V|\right)} \tag{6.19}$$

The way we will interpret $\Phi(G)$ is as follows. For any set S, if the neighbors of each vertex was chosen at random (total of $d|S|$ neighbors would have been chosen). The probability that each neighbor chosen in this way falls outside S, is $\left(|\overline{S}|/|V|\right)$. Hence the denominator is the expected number of crossing edges from S for a random graph. Thus, the conductance, intutively, denotes how much "random" the given graph is.

We first claim that conductance is related to edge expansion ratio.

CLAIM 6.5.2.

$$h(G) \leq \Phi(G) \leq 2h(G)$$

Proof. Note that the role of S and $\overline{S}$ are interchangable. Hence, without loss of generality, there is an S for which $|S| \leq \frac{n}{2}$ where the minimum is achieved in equation 6.19 and for that minimum, the factor $\frac{|\overline{S}|}{|V|}$ in the denominator is at least $\frac{1}{2}$ and at most 1. Hence the minimum achieved for equation 6.19 can be at most $2h(G)$ and at least at $h(G)$. ∎

Exercise 6.5.3. Prove Claim 6.5.2 for a general α.

6.6 Proof of Cheeger's Inequality: From Spectral to Combinatorial

We now prove the LHS of the Cheeger's inequality ((Theorem 6.5.1)). Because of the above claim, it suffices to prove that $\Phi(G) \geq 1 - \lambda_2(G)$.

Approach: We prove this by a general idea which is useful in other context too and hence we present it in the general form, before applying it here. Consider a minimization question with an objective function $\psi(z_1, z_2, \ldots, z_n)$ in terms of variables $z_1, z_2, \ldots z_n \in \{0, 1\}$. Let k be the optimum

value. Now suppose we optimize the same function without the constraint of the values being Boolean, that is $z_1, z_2, \ldots z_n \in \mathbb{R}$. Clearly the optimal value of ψ can only be smaller. It is this simple trick that we will apply in the context of the above two parameters too. We will write an objective function for which $\Phi(G)$ is the solution when the variables are restricted to $\{0,1\}$ and $1-\lambda_2(G)$ is the solution when the variables are relaxed to be arbitrary real numbers.

Writing $\Phi(G)$ as the result of a minimization: We start by writing $\Phi(G)$ as the result of an optimization problem. We encode $S \subseteq V$ as a bit string $x \in \{0,1\}^n$, and x_i denote the i^{th} bit of x.

$$\begin{aligned}
\Phi(G) &= \min_{\{\phi \subseteq S \subseteq [n]\}} \frac{|E(S,\overline{S})||V|}{d|S||\overline{S}|} && (6.20)\\
&= \min_{x \in \{0,1\}^n \setminus \{0^n,1^n\}} \frac{\frac{n}{2}\sum_{i,j} dA_{ij}(x_i - x_j)^2}{d(\sum_i x_i)(n - \sum_i x_i)} && (6.21)\\
&= \min_{x \in \{0,1\}^n \setminus \{0^n,1^n\}} \frac{n\sum_{i,j} A_{ij}(x_i - x_j)^2}{2(\sum_i x_i)(n - \sum_i x_i)} && (6.22)\\
&= \min_{x \in \{0,1\}^n \setminus \{0^n,1^n\}} \frac{n\sum_{i,j} A_{ij}(x_i - x_j)^2}{\sum_{i,j}(x_i - x_j)^2} && (6.23)\\
&\geq \min_{x \in \mathbb{R}^n \setminus \{0^n,1^n\}} \frac{n\sum_{i,j} A_{ij}(x_i - x_j)^2}{\sum_{i,j}(x_i - x_j)^2} && (6.24)
\end{aligned}$$

Now we decompose the vector x as: $x = \alpha \overrightarrow{\mathbf{1}} + w$ where $w \perp \overrightarrow{\mathbf{1}}$. In the above expression, noting that $x_i - x_j = w_i - w_j$, we can restrict the optimization to $x \perp \overrightarrow{\mathbf{1}}$. [16]

$$\begin{aligned}
\Phi(G) &\geq \min_{\substack{x \in \mathbb{R}^n \setminus \{0^n,1^n\} \\ x \perp \overrightarrow{1}}} \frac{n\sum_{i,j} A_{ij}(x_i - x_j)^2}{\sum_{i,j}(x_i - x_j)^2} && (6.25)\\
&\geq \min_{\substack{x \in \mathbb{R}^n \setminus \{0^n,1^n\} \\ x \perp \overrightarrow{1}}} \frac{n\sum_{i,j} A_{ij}(x_i - x_j)^2}{2x^T x - 2\sum_{i,j} x_i x_j} && (6.26)\\
&= \min_{\substack{x \in \mathbb{R}^n \setminus \{0^n,1^n\} \\ x \perp \overrightarrow{1}}} \frac{n\sum_{i,j} A_{ij}(x_i - x_j)^2}{2x^T x} && (6.27)
\end{aligned}$$

Writing $\lambda_2(G)$ as the result of a maximization: A natural starting point is the following expression of top two eigen values:

LEMMA 6.6.1 (**Courant-Fischer Formula for $\lambda_2(G)$**). *If $\lambda_2(G)$ is the second largest eigen value of the normalized adjacency matrix of a graph G: Then,*

$$\lambda_2 = \max_{\substack{x \in \mathbb{R} \setminus \{0\} \\ x \perp 1}} \frac{x^T A x}{x^T x} \tag{6.28}$$

[16] A consequence is that $\sum_{ij} x_i x_j = \sum_i x_i(\sum_j x_j) = \sum_i x_i(\langle x, \overrightarrow{1}\rangle) = 0$.

Proof. In fact, we start by deriving a similar formula for $\lambda_1(G)$ first and then adapt it to $\lambda_2(G)$. We prove:

$$\lambda_1 = \max_{x \in \mathbb{R} \setminus \{0\}} \frac{x^T A x}{x^T x} \tag{6.29}$$

We will estimate the numerator and denominator separately. We start with an orthonormal basis $v_1, v_2, \ldots v_n$ of $\mathbb{R}^n$ such that v_i is an eigen vector of λ_i. Any vector x can be expressed as $x = \sum_i \alpha_i v_i$ where $\alpha_i = \langle x, v_i \rangle$.

$$x^T A x = \left(\sum_{i=1}^n \alpha_i v_i\right)^T A \left(\sum_{i=1}^n \alpha_i v_i\right) = \left(\sum_{i=1}^n \alpha_i v_i\right)^T \left(\sum_{i=1}^n \alpha_i \lambda_i v_i\right) = \sum_{i=1}^n \lambda_i \alpha_i^2 \leq \lambda_1 \sum_{i=1}^n \alpha_i^2$$

$$x^T x = \left(\sum_{i=1}^n \alpha_i v_i\right)^T \left(\sum_{i=1}^n \alpha_i v_i\right) = \sum_{i=1}^n \alpha_i^2$$

Dividing the two and taking max over all $x \in \mathbb{R}^n \setminus \{0\}$, we have proved Equation 6.29 (Equality follows from the fact that x can also be $\mathbf{1}$). The formula for λ_2 follows if we restrict ourselves to $x \perp \mathbf{1}$, $\alpha_1 = 0$. Thus, $\forall x \in \mathbb{R}^n \setminus \{0\}$, $x^T A x \leq \lambda_2 \sum_{i=1}^n \alpha_i^2$ (and considering the case of x as the eigen vector corresponding to λ_2 establishes equality and hence Equation 6.28. ∎

Now we will just modify the Courant-Fischer formula to see it as the optimization of the same objective function as in Equation 6.27. We use the lemma, whose proof is left as an exercise.

Exercise 6.6.2. If A is the normalized adjacency matrix of G and $x \in \mathbb{R}^n \setminus \{0\}$:

$$\sum_{i,j} A_{ij}(x_i - x_j)^2 = 2x^T x - 2x^T A x$$

Applying this, $x^T A x = x^T x - \frac{1}{2} \sum_{i,j} A_{ij}(x_i - x_j)^2$.

$$\lambda_2 = \max_{\substack{x \in \mathbb{R}^n \setminus \{0^n, 1^n\} \\ x \perp \vec{1}}} \frac{x^T x - \frac{1}{2} \sum_{i,j} A_{ij}(x_i - x_j)^2}{x^T x} = 1 - \min_{\substack{x \in \mathbb{R}^n \setminus \{0^n, 1^n\} \\ x \perp \vec{1}}} \frac{\sum_{i,j} A_{ij}(x_i - x_j)^2}{2x^T x}$$

This matches with the objective function in Equation 6.27. Thus, $\Phi(G) \geq 1 - \lambda_2(G)$.

6.7 Proof of Cheeger's Inequality: From Combinatorial to Spectral

3: Jayalal says: Todo - Write down this proof

Exercise 6.7.1 (See Problem Set 3 (Problem 4))**.** In this problem, we will apply Cheeger's inequality and also show that it is tight.

(a) Let G be a complete graph on n vertices. Show that $h(G) \approx \frac{1}{2}$. Verify Cheeger's inequality.

(b) Let G be a cycle on n vertices. Compute λ_2 and $h(G)$ asymptotically and compare.

(c) Consider the graph G with 2^n vertices labelled with strings in $\{0,1\}^n$. The edges of G are as follows - two vertices to be adjacent if the corresponding strings differ by one bit flip. Show that $\lambda_2 = 1 - \frac{1}{n}$. Hence conclude that $h(G) \geq \frac{1}{2n}$. Derive an upper bound for $h(G)$ and compare.

6.8 Expander Mixing Lemma

We close this lecture by proving the expander mixing lemma, which is the reason expander graphs are thought of as having properties of random graphs yet being explicitly constructible. Recall the definition and interpretation of conductance of a graph that we saw in the last lecture, which measures, as a ration how much the cut of $E(S, \overline{S})$ is, compared to the expected number of edges across the sets if the graph (and hence the edges) were chosen at random. In this, we bound this in an additive fashion:

LEMMA 6.8.1 (**Expander Mixing Lemma**). *Let G be a d-regular graph with n vertices and let $\lambda_2 < 1$ be the second largest eigen value of the normalized adjacency matrix of G. Then, for any $S, T \subseteq [n]$:*

$$\left| |E(S,T)| - \frac{d|S||T|}{n} \right| \leq d\lambda_2 \left(\sqrt{|S||T|} \right)$$

REMARK 6.8.2. This lemma can be interpreted as follows. For any $S, T \subseteq V$, the expected number of edges between them if the edges are chosen at random is the second term. Indeed, the probability that a randomly chosen vertex is in the set T is $(|T|/n)$. Since there $d|S|$ vertices being chosen (as the other end point of edges where one endpoint is in S). Hence the expected number of edges that cross from S to T is $\frac{d|S||T|}{n}$.

Proof. We will again use the orthonormal basis of $\mathbb{R}^n : v_1, v_2, \ldots v_n$ such that $Av_i = \lambda v_i$ for each i. Consider the characterestic vectors of the two sets S and T as 1_S and 1_T. Express each of them in terms of the basis:

$$1_S = \sum_i \alpha_i v_i \text{ where } \alpha_i = \langle 1_S, v_i \rangle \quad \text{and} \quad 1_T = \sum_i \beta_j v_j \text{ where } \beta_i = \langle 1_T, v_i \rangle$$

Note that : $v_1 = (\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}, \ldots, \frac{1}{\sqrt{n}})$ and hence $\alpha_1 = \frac{|S|}{\sqrt{n}}$ and $\beta_1 = \frac{|T|}{\sqrt{n}}$

$$\begin{aligned} |E(S,T)| = 1_S^t dA 1_T &= \left(\sum_i \alpha_i v_i \right) dA \left(\sum_i \beta_j v_j \right) = d\alpha_1\beta_1 + d\sum_{i=2}^{n} \lambda_i \alpha_i \beta_i \\ &\leq d\alpha_1\beta_1 + d\lambda_2 \sum_{i=2}^{n} \alpha_i\beta_i \\ &\leq \frac{d|S||T|}{n} + d\lambda_2 \left(\sum_{i=2}^{n} \alpha_i\beta_i \right) \end{aligned}$$

Hence :

$$\left||E(S,T)| - \frac{d|S||T|}{n}\right| \leq d\lambda_2 \left(\sum_{i=2}^{n} \alpha_i\beta_i\right) \leq d\lambda_2 \left(\sum_{i=1}^{n} \alpha_i\beta_i\right) \leq d\lambda_2\|\alpha\|\|\beta\|$$

Note that

$$||\alpha|| = \sqrt{\sum_i \alpha_i^2} = \sqrt{\sum_i \langle 1_S, v_i\rangle^2} = ||1_S||$$

Hence the above inequality gives:

$$\left||E(S,T)| - \frac{d|S||T|}{n}\right| \leq d\lambda_2\|1_S\| \times \|1_T\| \leq d\lambda_2\sqrt{|S||T|}$$

□

Curiosity 6.8.3 (Converse of Expander Mixing Lemma). Expander Mixing Lemma (Lemma 6.8.1) says that the edges acorss S and T for any two subsets of matrices behaves like that of the random graphs. In fact, even the converse is also true. For any two subsets of vertices if the density of the subsets of matrices behaves like that of random graphs, then it must necessarily have a good spectral gap.

In fact, the converse of Expander Mixing Lemma (Lemma 6.8.1) is also known [Bilu and Linial, 2006]. The statement is as follows: Let G be a d-regular graph and suppose that

$$\left||E(S,T)| - \frac{d|S||T|}{n}\right| \leq d\delta\left(\sqrt{|S||T|}\right)$$

then $\lambda_2(G)$ is $O\left(\delta + \delta\log\left(d/\delta\right)\right)$.

Curiosity 6.8.4 (Ramanujan Graphs). Ramanujan graph, is a regular graph whose spectral gap is almost as large as possible. Such graphs are excellent spectral expanders. The complete graph K_{d+1} has spectrum $d, -1, -1 \ldots -1$ and thus $\lambda_2(K_{d+1}) = d$ and hence is a Ramanujan graph for every $d > 1$. The complete bipartite graph $K_{d,d}$ has spectrum $d, 0, 0, \ldots 0, -d$ and hence is a bipartite Ramanujan graph for every d. [Lubotzky et al., 1988] showed[17] how to construct an infinite family of $(p+1)$-regular Ramanujan graphs, whenever p is a prime number and $p \equiv 1 \mod 4$. This was extended for any prime power later.See [Murty, 2003] for a survey. It is still an open problem whether there are infinitely many d-regular (non-bipartite) Ramanujan graphs for any $d \geq 3$.

Exercise 6.8.5 (See Problem Set 3 (Problem 5))**.** Using techniques similar to what we used for mixing lemma, prove the following stronger version of the mixing lemma. Let G be a d-regular graph with n vertices and let $\lambda_2 < 1$ be the second largest eigen value of the normalized adjacency matrix of G. Then, for any $S, T \subseteq [n]$:

$$\left||E(S,T)| - \frac{d|S||T|}{n}\right| \leq d\lambda_2\left(\sqrt{|S||T|\left(1-\frac{|S|}{n}\right)\left(1-\frac{|T|}{n}\right)}\right)$$

[17] The proof uses what is called Ramanujan conjecture, which led to the name of Ramanujan graphs.

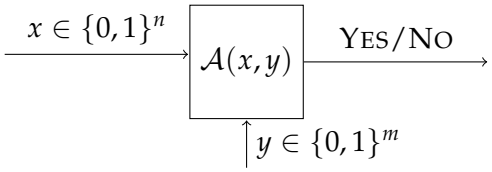
Efficient Error Reduction using Expander Graphs

Lecturer : Jayalal Sarma

WEEK

7

We provide two approaches to error reduction of algorithms using expander graphs. We fix some notations first. A randomized algorithm $\mathcal{A}$ on input x runs in time $t(n)$ (where $n = |x|$) and let $y \in \{0,1\}^{m(n)}$ be the concatenation of the unbiased coin toss experiement that the algorithm does during its execution. Notice that $m(n) \leq t(n)$ (we drop the n when it is not required explicitly). If the algorithm runs in polynomial time $t(n) \leq n^c$ for a constant c independent of n.



$$\forall x \in \{0,1\}^n, \Pr_{y \in \{0,1\}^m}[\mathcal{A}(x,y) \text{ is correct.}] \geq 1 - \epsilon$$

For an input $x \in 0,1^n$, let B be the set of random bits on which $\mathcal{A}$ makes an error.

$$B = \{y \in \{0,1\}^m \mid \mathcal{A}(x,y) \text{ errs}\}, \text{ and } |B| \leq \epsilon 2^m$$

We collect the different approaches that we have seen and will be seeing.

Repetition with Independent Random Bits: Repeat the experiment k times with independent random bits and accept the majority as the answer to be reported. This makes the new algorithm use km random bits, runs in time $kt(n)$ and achieve error bound of $\frac{1}{2^k}$.

Repetition with Expander Neighbors: We will achieve an error bound of $\frac{1}{\sqrt{k}}$ without spending any additional random bits and time $kt(n)$.

Repetition with Expander Walk: We will achieve an error bound of $\frac{1}{2^k}$ by using $m + O(k)$ random bits and time $kt(n)$.

Repetition with Pairwise Independence: Although not using expanders, we also use this context to introduce how basic notions of dependence can still be used in the context of amplification. We will show that if we use *pairwise independent* random variables, then it can still achieve error bounds of the form $\frac{1}{k}$ with only $O(\log n)$ additional random bits.

7.1 Approach 1 : Using Expander Mixing Lemma

The algorithm is as follows: let G be an $(2^m, d, \lambda)$ spectral expander. Let $N = 2^m$.

Note that the algorithm runs in time $O(dm)$ time and it does not use any extra random bits other than what the original algorithm does !. Now we show that it can still do some reasonable amplification.

Algorithm 7.14 ($\mathcal{A}'$) : input $x \in \{0,1\}^n$

1: Choose $y \in \{0,1\}^m$ uniformly at random.
2: Compute $N(y) = \{y_1, y_2, \ldots y_d\}$.
3: **for** `each` $i \in [k]$ **do**
4: If $\mathcal{A}(x, y_i)$ accepts, if so increment *count*
5: **end for**
6: If $[count > \frac{k}{2}]$ then output YES else output NO.
7: REJECT.

LEMMA 7.1.1. $\Pr[\mathcal{A}' \text{ errs }] \leq \frac{2\epsilon(\lambda_2)^2}{1-2\epsilon}$

Proof. Let B_x (we will drop subscript x from now on) denote the set of strings that makes the algorithm $\mathcal{A}$ err on input x. Thus we have: $|B| \leq \epsilon 2^m$. To analyse the algorithm $\mathcal{A}'$, we need to bound the probability that at least $\frac{d}{2}$ of the neighbors of the algorithm is in the set B. Let us define:

$$B' = \left\{ y \in \{0,1\}^m \ : \ |N(y) \cap B| \geq \frac{d}{2} \right\}$$

We need to show an upper bound for $\frac{|B'|}{2^m}$. This follows from the expander mixing lemma (Lemma 6.8.1) with $S = B'$ and $T = B$. This gives,

$$\left| |E(B,B')| - \frac{d|B||B'|}{2^m} \right| \leq d\lambda_2 \left(\sqrt{|B||B'|} \right)$$

Note that $|E(B,B')| \geq \frac{d}{2}|B'|$. And, $\frac{d|B||B'|}{2^m} \leq \epsilon d|B'|$. Since $\epsilon \leq \frac{1}{2}$, the first term in the LHS is larger.

$$\begin{aligned} \frac{d}{2}|B'| - d\epsilon|B'| &\leq d\lambda_2 \left(\sqrt{\epsilon 2^m |B'|} \right) \\ \frac{|B'|}{2^m} &\leq \frac{2\epsilon(\lambda_2)^2}{1-2\epsilon} \end{aligned}$$

As an example, if the original error probability is $\frac{1}{3}$, this gives a success probability of $2\lambda^2$. If we want the error bound to be less than $\frac{1}{m}$, we should choose a polynomial degree spectral expander with vertices as $\{0,1\}^m$ with $\lambda_2 \leq \frac{1}{2\sqrt{m}}$. ∎

REMARK 7.1.2. There are two curious aspects about the above proof. First of all, it looks like the error probability bound is independent of d - number of times the algorithm is repeated. Since d does not matter, can we take d as a constant? (This saves running time for us !). But then we need a spectral expander on n vertices with $\lambda_2 \leq \frac{1}{2\sqrt{\log |V|}}$. Can we have graphs with arbitrarily small second largest eigen value? Unfortunately, if we have degree to be small, then there is a constant fraction lower bound for λ_2. Hence our d has to grow with the number of vertices. We can afford to take $d = \text{poly}(\log |V|)$ in this case. So, in a general expander graph notation, we need n-vertex expander graphs of $\text{poly}(\log n)$ degree and $\lambda_2 \leq \frac{1}{\log n}$.

7.2 Approach 2 : Using Random Walk Mixing

We have seen the following amplification approach in Section 3.3. In this section, we show the formal treatment of the idea.

Algorithm 7.15 ($\mathcal{A}'$) : input $x \in \{0,1\}^n$

1: $count \leftarrow 0$.
2: Let $G(V,E)$ be an expander graph on 2^m vertices.
3: Choose a vertex $y_1 \in V$ uniformly at random. ▷ Uses m random bits
4: Starting at v perform a random walk for k steps in G. Let $y_1, y_1, y_2, \ldots y_k$ each in $\{0,1\}^m$ be the vertices representing the walk. ▷ Uses $O(k \log d)$ random bits.
5: **for** `each` $i \in [k]$ **do**
6: If $\mathcal{A}(x, y_i)$ accepts, if so increment *count*
7: **end for**
8: If $[count > \frac{k}{2}]$ then output YES else output NO.

As stated along with the algorithms description, the number of random bits used here is $m + O(k \log d)$, and since we can use a constant degree expander graph family, the number of random bits used is $m + O(k)$. The time taken by the algorithm is k times the original running time. We now bound the error probability of the algorithm.

LEMMA 7.2.1. $\Pr[\mathcal{A}' \text{ errs }] \leq 2^{-k}$

Proof. Again B will denote the set of strings that makes the algorithm $\mathcal{A}$ err on input x. Thus we have: $|B| \leq \epsilon 2^m$. To analyse the algorithm $\mathcal{A}'$, we need to bound the probability that at least $\frac{d}{2}$ of the neighbors of the algorithm is in the set B. Let $I = \{1, 2, \ldots k\}$ be the index set we use for denoting the repetition iteration. We need to estimate the probability:

$$\Pr\left[\begin{array}{c} \exists I \subseteq [k], |I| = k/2 \\ [\forall i \in I,\ y_i \in B] \end{array}\right]$$

where the probability is over the $m + O(k \log d)$ random bits used in the algorithm. As we have done before, the strategy is to remove the existential quantifier first (and then apply union bound later). Fix an $I \subseteq [k]$ of size $k/2$. We need to estimate the probability of the event $[\forall i \in I,\ y_i \in B]$. We will first do a simpler task - to estimate the probability when $I = [k]$.

Let us denote $\pi_1 \in \mathbb{R}^{2^m}$ to be the distribution corresponding to the first variable $y_1 \in \{0,1\}^m$. We first understand the case when $I = \{1\}$. We know that $\Pr[y_1 \in B] = \frac{|B|}{2^m}$. To express it in a form which we can use to iterate, define the operator $\Gamma : \mathbb{R}^{2^m} \to \mathbb{R}^{2^m}$ which for any $\phi \in \mathbb{R}^{2^m}$ projects to co-ordinates in B. More formally,

$$\Gamma(\pi)_i = \begin{cases} \pi_i & \text{if } i \in B \\ 0 & \text{otherwise} \end{cases}$$

For shorthand notation, we denote $\Gamma(\pi)$ as $\Gamma\pi$. Now $\Pr[y_1 \in B]$ can be written as $|\Gamma\pi|_1$. We need to understand when $I = \{1,2\}$, recall that the steps in the random walk is represented by

the linear transformation corresponding to the normalized adjacency matrix A.

$$\begin{aligned}\Pr[y_1 \in B \wedge y_2 \in B] &= \Pr[y_2 \in B \mid y_1 \in B]\Pr[y_1 \in B] \leq |\Gamma A \Gamma \pi_0|_1 \\ \Pr[y_1 \in B \wedge y_2 \in B \ldots y_k \in B] &\leq |(\Gamma A)^{k-1}\Gamma\pi_0|_1\end{aligned}$$

Thus, we need to upper bound $|(\Gamma A)^{k-1}\Gamma\pi_0|_1$. Note that $\Gamma \circ \Gamma = \Gamma$. Hence, $(\Gamma A)^{k-1}\Gamma \equiv (\Gamma A \Gamma)^{k-1}\Gamma$. We will replace $k-1$ with k in the expression from now on, for easy representation, but this is only going make the upper bound bigger, so it does not affect us.

Now here comes the advantage of moving to vectors. Instead of bounding the ℓ_1 norm, we will upper bound the ℓ_2 norm[18].

To do this, we use the following claim which indicates that this ℓ_2 norm actually gets closer and closer to $\epsilon\|\pi_1\|_2$ if k is large enough.

CLAIM 7.2.2. *For all k, and for all π:*

$$\left\|(\Gamma A \Gamma)^k \pi\right\|_2 \leq (\epsilon + \lambda)^k \|\pi\|_2$$

$$\left\|(\Gamma A^k \Gamma)\pi\right\|_2 \leq (\epsilon + \lambda^k)\|\pi\|_2$$

Proof. We prove the second inequality (which is needed in the next step too). Applying the second inequality for $k = 1$ repeatedly will yield the first inequality.

We need to prove :

$$\left\|\Gamma A^k(\Gamma\pi)\right\|_2 \leq (\epsilon + \lambda)\|\pi\|_2$$

The natural attack, given the above is to write $\Gamma\pi$ vector in terms of the orthonormal basis $v_1, v_2 \ldots v_{2^n}$ of the eigen spaces of the eigen values of the matrix A. Notice that since $\|v_1\|_2 = 1$, we have that the vector : $v_1 = \left(\frac{1}{\sqrt{2^m}}, \frac{1}{\sqrt{2^m}}, \ldots, \frac{1}{\sqrt{2^m}}\right)$.

Let $\Gamma\pi$ be written as $\Gamma\pi = \alpha_1 v_1 + w$ where $\alpha_1 = \langle \Gamma\pi, v_1\rangle$ and $w \perp v_1$. Hence, the term that we wanted to bound, $\left\|\Gamma A^k(\Gamma\pi)\right\|_2 \leq \alpha_1\left\|\Gamma A^k v_1\right\|_2 + \left\|\Gamma A^k w\right\|_2$. We bound the two terms in the summation separately.

SUB-CLAIM 1 : $\alpha_1\left\|\Gamma A^k v_1\right\|_2 \leq \epsilon\|\pi\|_2$:

$$\alpha_1\left\|\Gamma A^k v_1\right\|_2 = \alpha_1\|\Gamma v_1\|_2 = \alpha_1\sqrt{\sum_{i=1}^{|B|}\left(\frac{1}{\sqrt{2^m}}\right)^2} = \alpha_1\sqrt{\frac{|B|}{2^m}} \leq \alpha_1\sqrt{\epsilon}$$

Moreover,

$$\alpha_1 \leq \langle \Gamma\pi, v_1\rangle \leq \|\Gamma\pi\|_2\|v_1\|_2 \leq \|\Gamma\pi\|_2 \leq \sqrt{\sum_{i\in B}\pi_i^2} \leq \sqrt{\sum_{i\in B}\pi_i.\pi_i} \leq \sqrt{\sum_{i\in B}1.\pi_i} \leq \sqrt{\epsilon}\|\pi\|_2$$

[18]This is enough since, for any $v \in R^n$, $\frac{|v|_1}{\sqrt{n}} \leq \|v\|_2 \leq |v|_1$. To see first inequality which we are using, $|v|_1 = \sum_i |v_i| = \sum_i |v_i|.1 \leq \sqrt{\sum_i v_i^2 \sum_i 1^2} = \|v\|_2\sqrt{n}$ where the second last inequality follows from Cauchy-Schwartz inequality.

4: Jayalal says: Todo - the last inequality is clear when π is a uniform distribution, but that is not enough for us since we are applying it for arbitrary π as well. So this needs a fix.

SUB-CLAIM 2 : $\left\|\Gamma A^k w\right\|_2 \leq \lambda^k \|\pi\|_2$

Since Γ only inhibits certain co-ordinates, it can only diminish norms and this give $\left\|\Gamma A^k w\right\|_2 \leq \left\|A^k w\right\|_2 \leq (\lambda)^k \|w\|_2$. The last inequality follows because $\lambda = \max_{w \perp v_1} \frac{\|Aw\|_2}{\|w\|_2}$. Now we just need to use the fact that $\|w\|_2 \leq \|\Gamma\pi\|_2 \leq \|\pi\|_2$ and this gives $\left\|\Gamma A^k w\right\|_2 \leq \lambda^k \|\pi\|_2$ as needed.

The claimed bound in Claim 7.2.2 follows from the two subclaims. □

Using Claim 7.2.2 first part, we derive the probability bound that we wanted to estimate.

$$\begin{aligned}
\Pr[y_0 \in B \wedge y_1 \in B \ldots y_k \in B] &\leq |(\Gamma A \Gamma)^k \pi_0|_1 \leq \sqrt{2^m} \left\|(\Gamma A \Gamma)^k \pi_0\right\|_2 \\
&\leq \sqrt{2^m}(\epsilon + \lambda)^k \|\pi_0\|_2 \leq \sqrt{2^m}(\epsilon + \lambda)^k \frac{1}{\sqrt{2^m}} \leq (\epsilon + \lambda)^k
\end{aligned}$$

Now we understand how to estimate the membership in B for the whole of the index set, we take the next step to estimate it for a subset of $[k]$ of size $\frac{k}{2}$. Let $I \subseteq [k]$, $|I| = \frac{k}{2}$.

$$\Pr\left[\bigwedge_{i \in I} y_i \in B\right] \leq |(\Gamma A) A (\Gamma A)(A)(A)(\Gamma A)\Gamma \pi_0|_1$$

where, in the expression, we should see exactly $\frac{k}{2}$ (ΓA)s. We collect all of the As together and write:

$$\begin{aligned}
\Pr\left[\bigwedge_{i \in I} y_i \in B\right] &\leq |(\Gamma A^{k_1}\Gamma)(\Gamma A^{k_2}\Gamma)\ldots(\Gamma A^{k_t}\Gamma)\pi_0|_1 \\
&\leq \sqrt{2^m}\left\|(\Gamma A^{k_1}\Gamma)(\Gamma A^{k_2}\Gamma)\ldots(\Gamma A^{k_t}\Gamma)\pi_0\right\|_2 \\
&\leq \sqrt{2^m}(\epsilon + \lambda^{k_1})(\epsilon + \lambda^{k_2})\ldots(\epsilon + \lambda^{k_t})\frac{1}{\sqrt{2^m}} \\
&\leq (\epsilon + \lambda)^{(k/2)} \leq \frac{1}{4^k} \qquad \because \text{choose } \lambda \text{ such that } \epsilon + \lambda \leq \frac{1}{16}.
\end{aligned}$$

Now we apply the union bound as per our plan.

$$\begin{aligned}
\Pr\left[\begin{array}{c} \exists I \subseteq [k], |I| = k/2 \\ [\forall i \in I,\ y_i \in B] \end{array}\right] &\leq \left(\text{\# of Is of size } \frac{k}{2}\right) \times \Pr\left[\bigwedge_{i \in I} y_i \in B\right] \\
&\leq 2^k \times (\epsilon + \lambda)^{(k/2)} \leq \frac{1}{2^k}
\end{aligned}$$

Hence the proof.

□

We conclude this part by providing a table of the various amplification methods that we have seen so far.

Approach	Error bound	Random bits used
Trivial	$\frac{1}{2}(1-4\epsilon^2)^{k/2}$	km
Expander Neighborhood	$\frac{2\epsilon\lambda^2}{(1-2\epsilon)}$	m
Expander Walk	$2^k(\epsilon+\lambda)^{k/2}$	$m+k\log d$
Pairwise Independence	$\frac{4}{k(1-2\epsilon)^2}$	$2m$
t-wise Independence	$\frac{1}{k^{t/2}(1-2\epsilon)^t}$	tm

Construction of Spectral Expanders

Lecturer : Jayalal Sarma

WEEK

8

We now get into explicit construction of expanders. As discussed earlier, we rely on combinatorial constructions. Even though combinatorial, they go via spectral expandors. The outline of the approach is as follows. Suppose we have a graph G with spectrum as $1 = \lambda_1 \geq \lambda_2 \ldots \geq \lambda_n$. We want to amplify the gap between λ_1 and λ_2.

8.1 The Plan

We will build expanders from graphs which may not have expansion. To begin with, we have a small expansion for every regular graph. We prove the following theorem in the next section.

THEOREM 8.1.1 (**Every graph has noticable Spectral gap)**. *If G is a regular connected graph with self-loops at each vertex, then*

$$\lambda_2(G) \leq 1 - \frac{1}{4n^3}$$

Thus there is a spectral gap of $\frac{1}{4n^3}$. To convert this to an expander, a natural method to increase the gap is to power the eigen values - since $\lambda_1 = 1$ and $\lambda_2 < 1$. What opertation on the matrix will imply powering of the eigen values? Indeed, matrix powering. What combinatorial operation on the graph will imply a matrix powering of the adjacency matrix? We formally define this operation now.

DEFINITION 8.1.2 (**Graph Powering**). *If $G(V, E)$ is a d-regular digraph, then $G^k = (V, E')$ is a d^k-regular digraph on the same vertex set, every disinct walk of length k in G is replaced with a single edge in E'.*

The following observation follows from the fact that eigen values of the matrix A^k is exactly the square of the eigven values of A. This will incease the spectral gap.

LEMMA 8.1.3. *If G is an (n, d, λ) spectral expander graph, then G^k is an (n, d^k, λ^k) spectral expander.*

The question then is, if we want the spectral gap to be at least $\frac{1}{2}$, what should be the value of k? Indeed:

$$1 - \left(1 - \frac{1}{4n^3}\right)^k \geq \frac{1}{2} \qquad \text{which solves to } k \leq \mathsf{poly}(n)$$

But we should be worried about the degree. Although the new graph is regular, it is a $d^{\mathsf{poly}(n)}$-regular graph. However, we require a constant degree graph. Hence we need to decrease the degree. It is a delicate operation and it should not decrease the spectral gap too much. It turns out

that we can do this in a very precise way using some basic combinatorial tools which we describe in the next lecture.

8.2 Every Graph has a Noticable Spectral Gap

We prove Theorem 8.1.1 which shows that every graph with self loops has a small spectral gap already.

Proof of **Theorem 8.1.1.** Let $\epsilon = \frac{1}{2n^3}$. We will show that $\lambda_2(G) \leq 1 - \frac{\epsilon}{2}$. Consider a unit vector $x \perp \mathbf{1}$ vector in $\mathbb{R}^n$, it suffices to show that $\|Ax\|_2 \leq 1 - \frac{\epsilon}{2}$. Denote $y = Ax$. We need to show that $\|y\|_2 \leq 1 - \frac{\epsilon}{2}$.
We can simplify the target. Imagine that, $\|y\|_2 > 1 - \frac{\epsilon}{2}$. Then, $\|y\|_2^2 > \left(1 - \frac{\epsilon}{2}\right)^2 = 1 - \epsilon + \frac{\epsilon^2}{4} > 1 - \epsilon$. Hence it suffices to prove that $\|y\|_2^2 \leq 1 - \epsilon$. We view this as:

$$\|x\|_2^2 - \|y\|_2^2 \geq \epsilon$$

This says that Ax "crunches" the vector x since the difference between the norms of x and Ax is high. We will reinterpret the LHS of the above equvation in the following way. We use the fact that $\|y\|_2^2 = \langle Ax, y\rangle$.

$$\begin{aligned}
\|x\|_2^2 - \|y\|_2^2 &= \|x\|_2^2 - 2\langle Ax, y\rangle + \|y\|_2^2 \\
&= \sum_j \left(\sum_i A_{ij} x_j^2\right) - 2\sum_{ij}\left(\sum_j A_{ij} x_j\right) y_i + \sum_i \left(\sum_j A_{ij} y_i^2\right) \\
&= \sum_{ij} A_{ij} x_j^2 - \sum_{ij} A_{ij}(2x_j y_i) + \sum_{ij} A_{i,j} y_i^2 \\
&= \sum_{ij} A_{ij}(x_j^2 - 2x_j y_i + y_i^2) \\
&= \sum_{ij} A_{ij}(x_j - y_i)^2
\end{aligned}$$

Hence, it suffices to prove the following equation.

$$\sum_{i,j} A_{ij}(y_i - x_j)^2 \geq \epsilon \tag{8.30}$$

We show that there are terms in the above summation, which adds up to more than ϵ. This is sufficient since no term is negative in value.

Since x is a unit vector, there must exist an i such that $|x_i| \geq \frac{1}{\sqrt{n}}$. Since $x \perp 1$, there must exist a j such that x_i and x_j are of opposite signs and this implies that $|x_i - x_j|$ is at least $\frac{1}{\sqrt{n}}$. Notice that there must be a path between vertex i and vertex j in the graph G of length at most $n - 1$ (edges). By appropriately renaming it, let the path be $1, 2, \ldots n$ where the i-th vertex is renamed to 1 and

j-th vertex is renamed to n. With this renaming:

$$\begin{aligned}\frac{1}{\sqrt{n}} \leq |x_1 - x_n| &= |(x_1 - y_1) + (y_1 - x_2) + (x_2 - y_3) + (y_3 - x_4) \dots (y_n - x_n)| \\ &\leq |x_1 - y_1| + |y_1 - x_2| + |x_2 - y_3| + \dots |y_n - x_n| \\ &\leq \sqrt{2n}\left(\sqrt{(x_1 - y_1)^2 + (y_1 - x_2)^2 + (x_2 - y_3)^2 + \dots (y_n - x_n)^2}\right)\end{aligned}$$

The last inequality is using the relationship between ℓ_1 and ℓ_2 norm of vectors[19]. This implies that:

$$(x_1 - y_1)^2 + (y_1 - x_2)^2 + (x_2 - y_3)^2 + \dots (y_n - x_n)^2 \geq \frac{1}{2n}$$

Notice that each of the terms (in RHS) is in the above equation are positive and they appear in the RHS of the Equation 8.30 with a multiplication factor of $\frac{1}{d}$ (which is the entry[20] A_{ij}). Hence the above lower bound should imply a lower bound for Equationeqn:weak-exp as well. Since $d \leq n$.

$$\sum_{i,j} A_{ij}(y_i - x_j)^2 \geq \frac{1}{2dn} \geq \frac{1}{2n^3}$$

This implies the theorem. □

REMARK 8.2.1. The above theorem can be improved on the parameter side by applying a better upper bound on the diameter. Use the fact that between i and j there will be a path of length $\frac{3n}{d+1}$. (Prove this!) This will improve the spectral gap to $\frac{1}{12n^2}$. Another improvement known is the proof of the claim when it is not bipartite but does not have self-loops.

8.3 Amplifying the Spectral Gap : Power Product

As outlined in the beginning of this lecture, we defined graph operations which implies the required changes in the spectral gap. We start with the following.

DEFINITION 8.3.1 (**Graph Powering**). *Given a graph G, the k-th power of G, denoted by G^k is defined on V itself where there is an edge between two vertices v and w for every path of length k in G. Described in terms of adjacency matrix, the adjacency matrix of G^k is A^k where A is the adjacency matrix of G.*

THEOREM 8.3.2. *If G is an (n, d, λ) spectral expander, then G^k is an (n, d^k, λ^k) spectral expander.*

As mentioned before this is good for the spectral gap amplification, but it is not good for maintaining constant degree. So we need a method for reducing the degree of a graph, without reducing the spectral gap by much. We will introduce this in the next section.

[19] Ineed, it is known that for any vector $x \in \mathbb{R}^n$, $\frac{\|x\|_1}{\sqrt{n}} \leq \|x\|_2 \leq \|x\|_1$

[20] Notice that since G has selfloops, the terms of the form $x_i - y_i$ also appears.

8.4 Increasing the Number of Vertices : Tensor Product

We will discuss a graph operation which increases the number of vertices drastically, but at the same time does not affect the spectral gap much. The operation is called graph tensoring. It is exactly emulating the matrix tensoring, which we review first. Let A be an $n \times n$ matrix and B be an $m \times m$ matrix as follows.

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} \qquad B = \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \vdots & \\ b_{n1} & b_{n2} & \dots & b_{mm} \end{bmatrix}$$

$$A \otimes B = \begin{vmatrix} a_{11}\begin{bmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \vdots & \\ b_{n1} & b_{n2} & \dots & b_{mm} \end{bmatrix} & a_{12}\begin{bmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \vdots & \\ b_{n1} & b_{n2} & \dots & b_{mm} \end{bmatrix} & \dots & a_{1n}\begin{bmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \vdots & \\ b_{n1} & b_{n2} & \dots & b_{mm} \end{bmatrix} \\ \vdots & \vdots & \vdots & \\ a_{n1}\begin{bmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \vdots & \\ b_{n1} & b_{n2} & \dots & b_{mm} \end{bmatrix} & a_{n2}\begin{bmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \vdots & \\ b_{n1} & b_{n2} & \dots & b_{mm} \end{bmatrix} & \dots & a_{nn}\begin{bmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \vdots & \\ b_{n1} & b_{n2} & \dots & b_{mm} \end{bmatrix} \end{vmatrix}_{mn \times mn}$$

What happens to the eigen values? We leave this as the following exercise.

Exercise 8.4.1. If A has eigen values $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$, and B has eigen values $\lambda'_1 \geq \lambda'_2 \geq \ldots \geq \lambda'_m$, then $A \otimes B$ has eigen values as :

$$\{\lambda_i \lambda'_j \mid 1 \leq i \leq n, 1 \leq j \leq m\}$$

Now we ask the question, what combinatorial graph operation will have the above effect on the adjacency matrices of the graphs?

DEFINITION 8.4.2 (**Tensor Product of Graphs**). *If G is an (n, d, λ) spectral expander and H is an (n', d', λ') spectral expander, then the tensor product graph $G \otimes H$ is a graph on nn' vertices defined as follows:*

H replaces each vertex of G : *For every vertex x of G, the graph $G \otimes H$ has a copy of H (with only vertices). We call this the cluster at x, denoted by H_x.*

Edges Across Copies of H : *For each edge (x, y) in G, we place a bipartite version of the edges in H across the clusters H_x and H_y.*

The above statement implies that the second largest eigen value of $A \otimes B$ is $\max\{\lambda_2(G), \lambda_2(H)\}$ which we record as the following lemma.

LEMMA 8.4.3. *If G is an (n, d, λ) spectral expander and H is an (n', d', λ') spectral expander, then $G \otimes H$ is an $(nn', dd', \max\{\lambda, \lambda'\})$ spectral expander.*

8.5 Reducing to Constant Degree : Replacement Product

The replacement product will be used to reduce the degree of the graph without decreasing the spectral gap by much (or equivalently without increasing λ_2 by much). Let G be an (n, D, λ) where D is presumably very large. We take a graph H which is $(D, 2d, \delta)$ spectral expander. We formally define the replacement product as below.

DEFINITION 8.5.1 **(Replacement Product)**. *G is an (n, D)-graph and H is a (D, d)-graph*[21]*, then the replacement product $G \circledR H$ is defined as follows: Fix an ordering of vertices in G and H.*

H **replaces each vertex of** *G* **:** *For every vertex x of G, the graph $G \circledR H$ has a copy of H. We call this the cluster at x.*

Edges Across Copies of *H* **:** *For each edge (x, y) in G where y is the i-th neighbor of x and x is the j-th neighbor of y - , we place d parallel edges between the i-th vertex in the copy of H that replaces x and j-th vertex in the copy of H that replaces y.*

Thus for any vertex, the degree is exactly d within the same copy of H and there are d parallel edges to another vertex in another copy of G. Hence the degree of the resulting graph is exactly $2d$. Intuitively, we should expect the resulting graph to be still a good expander - because random walk is still set to mix almost equally fast (this is the reason we put in d parallel edges between two vertices in G in two copies of H. Hence the resulting graph should be a reasonably good expander. This can be proved in both worlds - both as a spectral expander and as an edge expander. We present these in the next two subsections.

8.5.1 Effect of Replacement Product : Spectral View

We now prove how good the graph $G \circledR H$ is, as a spectral expander, if G and H are good spectral expanders. The following lemma makes this precise.

LEMMA 8.5.2. ***Spectral Expansion in Replacement Product*** *If G is an $(n, D, 1-\epsilon)$ spectral expander and H is an $(D, 2d, 1-\delta)$ spectral expander, then $G \circledR H$ is a $(nD, 2d, 1 - \frac{\epsilon\delta^2}{24})$ spectral expander.*

Proof. The degree bounds follow directly from the definition of the product opertation. We will show that $\lambda_2((G \circledR H)^3) \leq 1 - \frac{\epsilon\delta^2}{8}$. This implies the claimed bound[22].

Step 1 : Understanding the Adjacency Matrix of the Product Graph: We express the adjacency matrix of the product graph using tensor product. Let A and B be the normalized adjacency matrix corresponding to the graph G and H respectively. Let R be the normalized adjacency matrix ($nD \times nD$ matrix) corresponding to the replacement product graph $G \circledR H$. We first express R in

[21] We are dropping the spectral gap in this since it is not relevant for the combinatorial definition of the replacement product.

[22] $\lambda_2((G \circledR H)) \leq \sqrt[3]{\lambda_2((G \circledR H)^3)} \leq \left(1 - \frac{\epsilon\delta^2}{8}\right)^{1/3} \leq \left(1 - \frac{\epsilon\delta^2}{24}\right)$

terms of A and B. If we ignore the edges across different clusters in the graph $G \circledR H$, then the adjacency matrix is obtained by $I_n \otimes B$.

How do we get the edges across the clusters? If P is the $nD \times nD$ permutation matrix corresponding to the rotation map of the graph. That is, the entry $P[(x,i),(y,j)] = 1$ if the y is the i-th neighbor of x and x is the j-th neighbor of y.

$$R = \frac{1}{2}(I_n \otimes B) + \frac{1}{2}P \tag{8.31}$$

$$(I_n \otimes J_D)P(I_n \otimes J_D) = A \otimes J_D \tag{8.32}$$

Step 2 : A Matrix Decomposition Lemma For this, we need the definition of matrix norm. If A is an $n \times n$, then $\|A\|$ is the maximum number α such that $\|Av\|_2 \leq \alpha\|v\|_2$ for every $v \in \mathbb{R}^n$. The norm of a matrix is larger than all the eigen values. For the largest eigen value λ, choose a unit eigen vector v,

$$\|A\| \geq \|Av\|_2 = \|\lambda v\|_2 = |\lambda|\|v\|_2 = |\lambda|$$

A doubly stochastic matrix also has a decomposition into "all-1s matrix" and the "rest" (with norm < 1). We also will state a linear algebraic decomposition lemma.

LEMMA 8.5.3. *The normalized adj. matrix of an (n,d,λ) graph can be written as : $A = (1-\lambda)J + \lambda K$ where J is an $n \times n$ matrix with all entries as $\frac{1}{n}$ and K has norm at most 1.*

One way to interpret the above lemma is that the random walk on the graph can be viewed as a convex combination of J (which is the random walk on complete graph (whose normalized adjacency matrix is J) and a random walk on a small norm (weighted) graph.

Proof. We claim $K = \frac{1}{\lambda}(A - (1-\lambda)J)$ satisfies the lemma. We prove that $\|K\| \leq 1$ first. Consider any $x \in \mathbb{R}^n$. We show that $\|Kx\|_2 \leq \|x\|_2$. Decompose $x = u + v$ where $w \perp 1$ and $u = \alpha 1$. Hence $\|u\|_2^2 + \|v\|_2^2 = \|x\|_2^2$. We want to estimate $\|Kx\|_2^2 = \|Ku + Kv\|_2^2$.

To understand Ku: We write $Ku = \frac{1}{\lambda}(Au - (1-\lambda)Ju))$. Since $Ju = u$ and $Au = u$, $Ku = u$.
To understand Kv: Using the fact that $v \perp 1$, $Kv = \frac{1}{\lambda}(Av - (1-\lambda)Jv)) = \frac{1}{\lambda}Av$ since $Jv = 0$.
Estimating $\|Kx\|_2^2$: Note that $Av \perp 1$ (since $\langle Av, 1\rangle = \langle v, A1\rangle = \langle w, 1\rangle = 0$). Thus, $Kv \perp Ku$.

$$\begin{aligned} \|Kx\|_2^2 &= \|Ku + Kv\|_2^2 = \|Ku\|_2^2 + \|Kv\|_2^2 \\ &= \|u\|_2^2 + \frac{1}{\lambda}\|Av\|_2^2 \leq \|u\|_2^2 + \|v\|_2^2 = \|x\|_2^2 \end{aligned}$$

□

Step 3: Bounding the norms: Applying the lemma to the two adjacency matrices : we have that $B = \delta J_D + (1-\delta)B'$ where B' has norm ≤ 1. Substituting this into Equation 8.32, we get that:

$$\begin{aligned} R &= \frac{1}{2}\left(I_n \otimes (\delta J_D + (1-\delta)B')\right) + \frac{1}{2}P \\ &= \left[\frac{\delta}{2}(I_n \otimes J_D) + \frac{(1-\delta)}{2}(I_n \otimes B') + \frac{1}{2}P\right] \end{aligned}$$

To get to the normalized adjacency matrix of $(G \circledR H)^3$. Consider R^3 and consider the terms appearing in the product. $(I_n \otimes J_D)$ and $(J_n \otimes I_D)$ commutes with each other and $(I_n \otimes J_D) = J_{nD}$, we have:

$$\begin{aligned} R^3 &= \frac{\delta^2}{8}\left[(I_n \otimes J_D)P(I_n \otimes J_D)\right] + \left(1 - \frac{\delta^2}{8}\right)R' \qquad \text{where } R' \text{ is a matrix of norm at most 1.} \\ &= \left(\frac{\delta^2}{8}\right)(A \otimes J_D) + \left(1 - \frac{\delta^2}{8}\right)R' \end{aligned}$$

To find out the second largest eigen value of R^3. The eigen values of the first term is $\frac{\delta^2}{8}$ times the eigen value of $A \otimes J_D$. The latter is exactly the second largest eigen value of A. Since $\|R'\| \leq 1$, this gives $\lambda_2((G \circledR H)^3) \leq 1 - \frac{\epsilon\delta^2}{8}$. Hence the proof. □

8.5.2 Effect of Replacement Product : Combinatorial View

We now present a direct combinatorial argument that the replacement product of two edge expanders gives a reasonably good edge expander with reduced degree. The construction of the graph is same as what we described in the beginning of the section. We prove the following theorem.

THEOREM 8.5.4 (**Edge Expansion in Replacement Product**). *If G is (n, D, δ_1) edge expander and H is (D, d, δ_2) edge expander, then $G \circledR H$ is $(nD, 2d, \frac{1}{80}\delta_1^2\delta_2)$ edge expander.*

Proof. For any $S \subseteq [nD]$, a set of vertices in $G \circledR H$ such that $|X| \leq \frac{nD}{2}$, we should prove,

$$\left|E(S, \overline{S})\right| \geq \frac{2d}{80}\left(\delta_1^2\delta_2|S|\right)$$

The idea of the proof is quite straightfoward. We will directly count the number of edges. Recall that the replacement product, places copies of H for each vertex in G. We call $H_1, H_2, \ldots H_n$ to be these clusters. Given $S \subseteq V$, $|S| \leq \frac{nD}{2}$, we want to count $E(S, \overline{S})$. Intuitively, there are "inter-cluster edges" and "intra-cluster edges" going out of S which we need to count separately. The former set should use the expansion of G and the latter should use the expansion of H. In a given cluster, the intra-cluster edges will be maximum if the number of vertices in S from the cluster is very close to half of the vertices in the cluster. Thus, we need to distinguish between clusters which have large intersection with S and others. For accounting this carefully, we will define the following sets.

Let $I \subseteq [n]$ be the index into the clusters where the intersection is small (and hence they will expand within their own cluster).

$$I = \left\{ i \in [n] \;\middle|\; |S \cap H_i| \leq \left(1 - \frac{\delta_1}{4}\right)D \right\}$$

This classifies the clusters into two kinds (with respect to the given S) - *sparse* (we will denote by $A_i = S \cap H_i$, $i \in I$) and *dense* clusters (we will denote them by $B_j = S \cap H_j$, $j \notin I$). By a slight

misuse of notation, we will denote by $\overline{A_i} = H_i \setminus A_i$ and $\overline{B_j} = H_j \setminus B_j$. Let,

$$A = \bigcup_{i \in I} A_i \qquad \overline{A} = \bigcup_{i \in I} \overline{A_i} \qquad B = \bigcup_{j \notin I} B_j \qquad \overline{B} = \bigcup_{j \notin I} \overline{B_j} \qquad S = A \bigcup B$$

Intuitively, there cannot be too many dense clusters since each dense cluster has to account for at least $\left(1 - \frac{\delta_1}{4}\right) D$ many vertices. This implies that

$$|\bar{I}| = |[n] \setminus I| \leq \frac{|B|}{(1 - (\delta_1/4))\, D} \leq \frac{(nD/2)}{(1 - (\delta_1/4))\, D} \leq \frac{n}{(2 - (\delta_1/2))} \leq \frac{2n}{3}$$

That means, at least $n/3$ of the clusters are sparse and at most $2n/3$ clusters are dense.

Contributors to the Cut $E(S, \overline{S})$: Based on the above classification, we have the following four different disjoint summands which contributes to $E(S, \overline{S})$ which is the sum of :

$$E\left(S, \overline{S}\right) \;=\; \left|E\left(\bigcup_{i \in I} A_i\,,\; \bigcup_{i \in I} \overline{A_i}\right)\right| + \left|E\left(\bigcup_{j \notin I} B_j\,,\; \bigcup_{j \notin I} \overline{B_j}\right)\right| + \left|E\left(\bigcup_{i \in I} A_i\,,\; \bigcup_{j \notin I} \overline{B_j}\right)\right| + \left|E\left(\bigcup_{j \notin I} B_j\,,\; \bigcup_{i \in I} \overline{A_i}\right)\right|$$

Since we need to show a lower bound for $|E(S, \overline{S})|$, we can count the approrpiate term in the above and prove a lower bound.

A first natural attempt is to consider the sparse clusters (the first summand in the above summation). Since H is an expander, they will have a lot of intra-cluster edges between S and $\overline{S}$. This works if there are many vertices in the sparse clusters in S. This gives rise to the following two cases.

Case 1 : Many sparse clusters - $|A| \geq \frac{1}{10}\delta_1 |S|$ More specifically, we claim the following, which also implies a lower bound on the first term in the above summation.

CLAIM 8.5.5 (**Intra-cluster Edges Within Sparse Clusters**). *For $i \in I$:*

$$|E(A_i, H_i \setminus A_i)| \geq \frac{1}{4}\delta_1 \delta_2 d |A_i|$$

Proof. We use the fact that H is a (D, d, δ_2)-edge expander. We have two cases.

Case 1: $|A_i| \leq \frac{D}{2}$**:** $|E(A_i, H_i \setminus A_i)| \geq \delta_2 d |A_i| \geq \frac{1}{4}\delta_1 \delta_2 d |A_i|$

Case 2: $|H_i \setminus A_i| \leq \frac{D}{2}$**:** we have that $|E(H_i \setminus A_i, A_i)| \geq \delta_2 d |H_i \setminus A_i| \geq \frac{1}{4}\delta_1 \delta_2 d D \geq \frac{1}{4}\delta_1 \delta_2 d |A_i|$

Recall that $A = \bigcup_{i \in I} A_i$. Since $|A| \geq \frac{1}{10}\delta_1 |S|$, then this implies the required bound.

$$|E(A_i, H_i \setminus A_i)| \geq \frac{1}{4}\delta_1 \delta_2 d |A_i| \geq \frac{1}{4}\delta_1 \delta_2 d \left(\frac{1}{10}\delta_1 |S|\right) \leq \frac{2d}{80}\left(\delta_1^2 \delta_2 |S|\right)$$

Case 2 : Many Dense Clusters - $|A| < \frac{1}{10}\delta_1 |S|$ **:** We have that $|B| \geq (1 - \frac{1}{10})\delta_1 |S| \geq \frac{9}{10}|S|$. In this case, we will lower bound the fourth term in Equation 8.33. More specifically, we claim the following :

CLAIM 8.5.6.

$$\left| E\left(\bigcup_{j \notin I} B_j \,,\, \bigcup_{i \in I} \overline{A_i} \right) \right| \geq \frac{1}{24} \delta_1 d |S|$$

Proof. We will translate the statement in terms of $|\bar{I}|$ first. More precisely, it suffices to prove that:

$$\left| E\left(\bigcup_{j \notin I} B_j \,,\, \bigcup_{i \in I} \overline{A_i} \right) \right| \geq \frac{1}{24} \delta_1 dD |\bar{I}| \tag{8.33}$$

This is sufficient because since each dense cluster can have at most D vertices from S, we know that $|\bar{I}| \geq \frac{|B|}{D} \geq \frac{9|S|}{10D} \geq \frac{|S|}{2D}$.

Now we prove Equation 8.33. We decompose the LHS of Equation 8.33.

$$\begin{aligned}
\left| E\left(\bigcup_{j \notin I} B_j \,,\, \bigcup_{i \in I} \overline{A_i} \right) \right| &= \left| E\left(\bigcup_{j \notin I} H_j \,,\, \bigcup_{i \in I} \overline{A_i} \right) \right| - \left| E\left(\bigcup_{j \notin I} \overline{B_j} \,,\, \bigcup_{i \in I} \overline{A_i} \right) \right| \\
&= \left| E\left(\bigcup_{j \notin I} H_j \,,\, \bigcup_{i \in I} H_i \right) \right| - \left| E\left(\bigcup_{j \notin I} H_j \,,\, \bigcup_{i \in I} A_i \right) \right| - \left| E\left(\bigcup_{j \notin I} \overline{B_j} \,,\, \bigcup_{i \in I} \overline{A_i} \right) \right|
\end{aligned}$$

We will lower bound the first term and upper bound the second and third terms.

Lower Bounding First Term: This is easier to bound since the term is exactly $d \left| E(I, \bar{I}) \right|$. Since $\left| \bar{I} \right|$ is small as a subset of vertices in G ($\left| \bar{I} \right| \leq \frac{2n}{3}$) and since G is an expander,

$$\left| E\left(\bigcup_{j \notin I} H_j \,,\, \bigcup_{i \in I} H_i \right) \right| = d \left| E(I, \bar{I}) \right| \geq \delta_1 dD \left| \bar{I} \right|$$

Upper Bounding Second Term: We will upper bound this crudely. Since $|A$ is small), and the edges coming out from any A vertex is at most d, we have that:

$$\left| E\left(\bigcup_{j \notin I} H_j \,,\, \bigcup_{i \in I} A_i \right) \right| \leq d|A| \leq \frac{\delta_1 Dd|S|}{10} \leq \frac{\delta_1 dD|\bar{I}|}{9}$$

where the last line follows because $|\bar{I}| \geq \frac{|B|}{D} \geq \frac{9|S|}{10D}$.

Upper Bounding Third Term:

$$\left| E\left(\bigcup_{j \notin I} \overline{B_j} \,,\, \bigcup_{i \in I} \overline{A_i} \right) \right| \leq \left| E\left(\bigcup_{j \notin I} \overline{B_j} \,,\, \bigcup_{i \in I} H_i \right) \right|$$

By definition of dense sets, $|B_j| \geq (1 - \frac{\delta_1}{4})D$. Hence $|H_j \setminus B_j| \leq \frac{\delta_1}{4} D$. This gives a bound

on the number of edges coming out of $H_j \setminus B_j$ is at most $\frac{\delta_1}{4} dD$. Thus,

$$\left| E\left(\bigcup_{j \notin I} \overline{B_j}\,,\ \bigcup_{i \in I} H_i \right) \right| \leq \frac{\delta_1}{4} dD|\bar{I}|$$

Thus the LHS in Equation 8.33 is: (note that $|\bar{I}|$

$$\begin{aligned} \left| E\left(\bigcup_{j \notin I} B_j\,,\ \bigcup_{i \in I} \overline{A_i} \right) \right| &\geq \delta_1 dD|\bar{I}| - \frac{\delta_1}{9} dD|\bar{I}| - \frac{\delta_1}{4} dD|\bar{I}| \\ &\geq \frac{1}{48}\delta_1(2d)D\left(\frac{9|S|}{2D}\right) \geq \frac{1}{80}\delta_1^2\delta_2(2d)|S| \end{aligned}$$

Hence the proof of claim 8.5.6. □

This completes the proof of Theorem 8.5.4 the combinatorial proof of expansion of the replacement product graph. □

8.6 Explicit Construction of Expanders

We now describe the explicit construction of expander graphs. We first comment on the notion of explicitness that we required. Note that we are interested in constructing a family of expander graphs $\{G_n\}_{n \geq 0}$ where each graph is d-regular for a constant d. The complexity of the family is hence described in terms of how efficiently these graphs can be described.

Weakly Explicit Expanders: There must be an algorithm which, given n in unary, output the adjacency matrix of G_n in the family, in time polynomial in n.

Strongly Explicit Expanders: There must be an algorithm for computing the rotation map of the graph G_n. Given n in binary, and a vertex label u, and a value $1 \leq i \leq d$, output the vertex v where v is the i-th neighbor of the vertex u in the graph G_n in the family, in time polynomial in $\log n$ (which is the input size).

Note that that the Marguilis and Gabber-Galil expanders that we described (see Lemma **??**) are strongly explicit by construction since we can explicitly write down the neighbors of the vertex in the order itself.

Construction: We now describe a construction of expander graphs (due to []) which will use the framework of graph products that we descirbed in this lecture. The idea is quite straight forward. We use powering for improving the spectral gap, which increases the degree. We use tensoring to increase the number of vertices, which again increases the degree. We use replacement product to reduce the degree of the graph without deteriorating the spectral gap by much. Do this iteratively. In the end, we must be getting a graph with a good spectral gap. By Cheeger's inequality, this must be a good edge expander. We now formally describe the construction.

- Let H be a $(D, \frac{d}{2}, \frac{1}{100})$ spectral expander graph where $D = d^{40}$. We will find such a graph by brute force search.

- Let G_1 be a $(D, d^{20}, \frac{1}{2})$ spectral expander graph. Again, we find this by brute force search. Choose d to be a large enough constant such that graphs H and G_1 exist.
- Define for every $k \geq 2$:

$$G_k = ((G_{k-1} \otimes G_{k-1}) \circledR H)^{20}$$

We list down properties of this construction.

Number of vertices: We claim that the graph G_k has at least 2^{2^k} vertices. To see this, if n_k is the number of vertices in G_k take $n_0 = 1$, and G_1 has D vertices. We have the recurrence, $n_k = (n_{k-1})^2 D$ since powering does not change the number of vertices. This gives, if $n_{k-1} = 2^{2^{k-1}}$

$$n_k \geq \left(2^{2^{k-1}}\right)^2 = 2^{2^k}$$

Thus, if we want to produce a graph on n vertices, then $k \in O(\log \log n)$.

Degree: The degree of the graph after replacement graph will be d, and that gets powered by the powering operation. Hence it becomes, d^{20} which is still a constant. The graph is regular too, by construction.

Spectral Expansion: Suppose $\lambda_2(G_{k-1}) \leq \frac{1}{3}$. After the tensor product, the λ_2 remains $\frac{1}{3}$ (see Lemma **??**). After the replacement product $\lambda_2 = 1 - \frac{1}{36}$. After powering, the resulting graph G_k has, $\lambda_2(G_k) = (1 - \frac{1}{36})^{20} \leq \frac{1}{3}$.

Explicitness: We claim that there is a $2^{O(k)}$-time algorithm that given a label of a vertex u in G_k and an index i, $1 \leq i \leq d^{20}$, outputs the i-th neighbor of u in the graph G_k. Since $k = O(\log \log n)$, this is $O(\log^c n)$, which is polynomial in the input size and meets the requirement for strong explicitness. This follows from the definition itself, by observing that to compute rotation map function of G_k, the algorithm will make 40 recursive calls to the rotation map of the graph G_{k-1}.

An alternative to the construction defined above is as follows. Define for every $k \geq 2$:

$$G_k = (G_{k-1} \otimes G_{k-1})^{20} \circledR H$$

In fact, this is more logical since powering is immediately followed by a replacement product. The analysis of the construction is similar.

REMARK 8.6.1. The disadvantage of the above construction is that it provides graphs only of size 2^{2^k} and that gives a large values of n for which we cannot construct expanders of size n. Ideally, we would like to construct expanders which are denser. For example, can we reduce this "sparsity" from double exponential to single exponential 2^k? This way, if we require a graph of size n, then there is a graph whose number of vertices is at most $2n$ (since between n and $2n$ there must be a power of 2). There is a variant of the above construction supplying a denser family of graphs that contains a graph with n vertices for every n that is a power of c, for some constant c. Noticing that we can transform an (n, d, λ) spectral expander graph to an (n', cd', λ) spectral expander graph for any $\frac{n}{c} \leq n' \leq n$ by merging a set of c vertices into one vertex, this gives a construction of expander graphs for every n.

Random Walks on Expanders, Reachability

Lecturer : Jayalal Sarma

WEEK

9

In this lecture, we make the statement "random walk in expanders mixes rapidly" precise and prove the same. We introduce it through the following running example.

9.1 Reachability Problem

The graph reachability problem takes input as a directed graph G and two vertices source (s) and desination (t) and asks to check if there is a path from s to t in G. Indeed, the trivial algorithm for the problem based on DFS or BFS, and the algorithm runs in time polynomial in n and uses $O(n)$ space of storage (for example, storing the *visited* array in a standard implementation. It is a natural question to ask whether there is an algorithm that can do better in terms of space keeping the efficiency in terms of time in tact.

Savitch[Savitch, 1970] showed the first (and the best known so far !) space complexity improvement for the problem. Savitch's algorithm runs in space $O(\log^2 n)$ but however, runs in time $O(n^{\log n})$. An question is to improve this algorithm further, either in terms of space or time. Indeed, the fundamental question in the area is whether or not there is an $O(\log n)$ space algorithm for the problem (any $O(\log n)$ space bounded algorithm can be bounded in time by $\mathsf{poly}(n)$). This question, while looks like a problem specific algorithm design challenge, captures on of the fundamental question in space complexity theory - the determinism vs non-determinism in the context of space - the NL vs L problem.

Reachability for restricted graph classes has been studied. A natural restriction is that of directed acyclic graphs. It turns out that this case is as hard as the general case. That is, we can reduce (in log space) a general graph reachability problem to an instance of the reachability problem where the graph is guaranteed to be acyclic.

A second natural restriction is that of undirected graphs which is the object of study in this lecture. We will first describe a simple randomized algorithm for the problem which is based on random walks on graphs. As for the heads up, in a break through result in 2004, Reingold[Reingold, 2008] designed a $O(\log n)$ space deterministic algorithm for reachability in undirected graphs. The algorithm is based on expanders and replacement product that we discussed in the last lecture.

9.2 Random Walk Based Algorithm for Reachability

We first describe the algorithm based on random walk. Let d be the degree of the given graph. The algorithm directly performs a random walk on the graph G starting from the source vertex s and

if t is found on the way, it accepts, or else, if walked long enough it declares that t is not reachable.

Algorithm 9.16 (REACH : input (G, s, t))

1: `curr` $= s$;
2: **for** `repeat the following for` ℓ `steps` **do** ▷ We will fix ℓ later to be poly(n)
3: If `curr` $= t$ then ACCEPT.
4: Choose a neighbor (`next`) of the vertex `curr` in G uniformly at random. ▷ $O(\log d)$ bits
5: `curr` = `next`.
6: **end for**
7: REJECT.

Now we analyse the algorithm. We analyse the space bounds first. The for loop takes a counter to implement it and it has count from 1 to ℓ. Since ℓ is going to be chosen as poly(n) this takes at most $O(\log n)$ bits to store the counter. Inside the loop, the variables stored are `next` and `curr` and the randbom bits generated. The first two takes $O(\log n)$ bits to store since the graph G has n vertices and the third takes $O(\log d)$ bits. Since $d \leq n$, the total space used inside the loop is at most $3 \log n$ bits and hence the entire algorithm runs in $O(\log n)$ space.

Now we get to correctness. The first observation is that if there is no path from s to t in the graph G, then the algorithm will never ACCEPT since it accepts only when it reaches t during the computation in Step 3. However, it may make an error when t actually has a path from s, when the random walk just goes somewhere else due to some unfortunate outcomes of the coin tosses. We would like to argue that the probability (over the random choices that the algorithm makes) that the algorithm indeed reaches is t is at least noticable, say at least $\frac{1}{n}$ (and then we will employ amplification techniques to make it better). Formally, we would like to prove:

CLAIM 9.2.1. *Let G be such that $s \rightsquigarrow t$ in G:*

$$\Pr[\text{ REACH } \textit{(G,s,t) accepts} \text{ }] \geq \frac{1}{n}$$

Clearly, this requires analysis of how random walk on the graph G, which we will do in the next section.

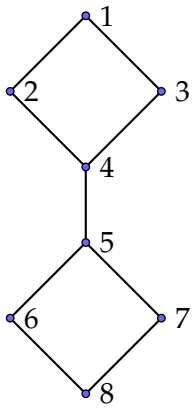
9.3 Convergence of Random Walks

Recall how we mathematically represented random walks in Section 6.4. Let X_i be the random variable denoting the vertex after the i^{th} step. Define the vector p_i, with entries $p_i[j] = \Pr[X_i = j]$. Hence we can rewrite the claim 9.2.1 as :

CLAIM 9.3.1. *Let G be such that $s \rightsquigarrow t$ in G:*

$$\Pr[X_\ell = t] = \Pr[\ p_\ell[t]\ ] \geq \frac{1}{n}$$

	1	2	3	4	5	6	7	8
p_0	1	0	0	0	0	0	0	0
p_1	0	0.5	0.5	0	0	0	0	0
p_2	0.5	0	0	0.5	0	0	0	0
p_3	0	0.42	0.42	0	0.17	0	0	0
$\vdots$			$\vdots$			$\vdots$		



$$p_{i+1}[j] = \Pr[X_{i+1} = j] = \sum_k \Pr[X_i = k] \Pr[\text{ } (k,j) \text{ edge is chosen }]$$

This implies that $p_{i+1} = Ap_i$.

Thus, the required claim follows from the following lemma.

LEMMA 9.3.2. *If A is the normalized adjacency matrix of an undirected graph and $p \in \mathbb{R}^n$ where p is a probability vector then,*

$$\left\| A^\ell p - u \right\|_2 \leq \lambda_2^\ell$$

where u is the vector $\left(\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n}\right)$.

We quickly apply the above lemma to prove the claim 9.3.1 and hence derive the value of ℓ in the algorithm. Recall from Theorem 8.1.1 that every connected graph with self loop on every vertex has a spectral gap of $\frac{1}{12n^2}$. Hence $\lambda_2 = 1 - \frac{1}{12n^2}$. We first ensure that the gap between $A^\ell p$ and u are small - formally, $\left\| A^\ell p - u \right\|_2 \leq \frac{1}{n^2}$. Thus, we choose an ℓ such that

$$\left(1 - \frac{1}{12n^2}\right)^\ell < \frac{1}{n^2} \quad \text{which happens if we choose } \ell = 24n^2 \log n$$

Thus, if we choose ℓ to be at least this large, $\left\| A^\ell p - u \right\|_2 \leq \frac{1}{n^2}$. If we denote $v = A^\ell p$, this says, $\sqrt{\sum_i (v_i - u_i)^2} \leq \frac{1}{n^2}$. Since each term is positive, for every i, $|v_i - u_i| \leq \frac{1}{n^2}$. The lowest value of v_i can only be $u_i - \frac{1}{n^2}$. Hence,

$$\forall i \in [n],\ (A^\ell p)_i \leq \frac{1}{n} - \frac{1}{n^2} \leq \frac{1}{2n}$$

Thus proving claim 9.3.1. We now provide the proof of Lemma 9.3.2 that we used.

Proof of Lemma 9.3.2. We need to understand $A^\ell p$. Decompose $p = \alpha u + p'$ where $p' \perp u$. Notice that sum of the components of p' is 0. Since p is a probability vector, its componenents should add up to exactly 1. Hence we can conclude that $\alpha = 1$. Thus, $p = u + p'$.

Now observe that $A^\ell p = A^\ell u + A^\ell p' = u + A^\ell p'$. Hence, $A^\ell p - u = A^\ell p'$. Using the fact that for any vector $x \in \mathbb{R}^n$, such that $x \perp 1$, $\|Ax\|_2 \leq \lambda_2 \|x\|_2$, we derive:

$$\left\| A^\ell p - u \right\|_2 = \left\| A^\ell p' \right\|_2 \leq \lambda^\ell \|p'\|_2$$

To upper bound $\|p'\|_2$, note that $\|p\|_2^2 = \|u\|_2^2 + \|p'\|_2^2$. This gives, $\|p'\|_2^2 \leq \|p\|_2^2 \leq |p|_1^2 \leq 1$. Hence the proof. ∎

9.4 Diameter of Expander Graphs

We now derive a nice property of expander graphs which will imply a deterministic algorithm for reachability which uses small space.

LEMMA 9.4.1. *For an $(n, d, \frac{1}{2}, \beta)$ vertex expander, the diameter of G is at most $O(\log(n))$.*

Proof. Let $u, v \in V$, Run a breadth first search from $S = \{u\}$ and count the number of vertices that we see in k distance from u. Clearly the set S will keep expanding to $(\beta d)^k |S| = (\beta d)^k$ in k steps. Choose the smallest k such that $(\beta d)^k > \frac{n}{2}$ which implies $k \in O(\frac{\log(\frac{n}{2})}{\log(\beta d)}) = O(\log n)$. Let T be the resulting set of vertices. Now, run the same process from v as well choosing $S' = \{v\}$ as the initial set to get to the set T'. Since the $|T|$ and $|T'|$ are more than $\frac{n}{2}$, we must have $T \cap T' \neq \phi$. This gives a path from u to v of distance at most $O(\log(n))$. Hence the diameter of the graph is at most $O(\log n)$. □

This gives a deterministic algorithm for reachability testing in undirected graphs G, where each component of the graph is degree at most d for each vertex[23]. Indeed, if s and t are provided, we can run a DFS only up to depth $O(\log n)$ remembering the path being explored currently by storing the index of each neighbor we followed of each vertex on the path. This gives $O(\log n \log d)$ space algorithm. When d is a constant, this gives $O(\log n)$ space algorithm as desired.

9.5 Expanderization Process

Given the above section, the following approach to solving reachability problem is quite natural : given (G, s, t) can we convert (G, s, t) to an expander (G', s', t') in such a way that: (1) conversion preserves reachability - that is, $s \rightsquigarrow t$ in G if and only if $s' \rightsquigarrow t'$ in G' (2) G' is a constant degree vertex expander (3) G' is constructible (implicitly in $O(\log n)$ space) from G. Indeed, each of these steps is quite challenging.

We directly describe the construction since we have developed all the required tools for it.

Step 1: Given the graph (G, s, t), reduce the degree of the graph to at most 3 by doing the following: for every vertex v with degree more than 3, replace v with a cycle of d vertices and connect each of the neighbors of v to the vertices of the cycle. Notice that this construction preserves reachability and does not introduce paths between the original vertices which originally were not connected. We can make the degree of each vertex to be exactly 3 by adding parallel edges.

Step 2: Add a self-loop to each vertex. The resulting graph is a 4-regular graph with a small spectral gap of $\frac{1}{12n^2}$ (see Theorem 8.1.1). By adding more self-loops we may assume that the graph is of degree d^{20} for some constant d that is sufficiently large so that there exists a $(d^{20}, d, 0.01)$-spectral expander graph H. Find H by bruteforce search.

[23] We can even assume that the graph is regular, by introducing multiple edges between two vertices where there is already and edge.

Step 3: Let $G_0 = G$ and inductively define

$$G_k = (G_{k-1} \circledR H)^{50}$$

Step 4: Run the deterministic algorithm for testing reachability in expander graphs for the graph G_k where $k = 10n \log n$. Without storing G_k explicitly (which we cannot since it can be of polynomial size).

There are multiple aspects of this constrution to be discussed.

Correctness: Observe that by doing replacement product and powering, we never introduce a spurious path between s and t if it did not exist in G already. This follows by definition.

Size of G_k. If size of G_k is n_k vertices, then the $n_k = n_{k-1}d^{50}$. Thus G_k has d^{50k} vertices. When $k = O(\log n)$, this is still a a polynomial size graph such that each vertex label is of length $O(\log n)$ bits.

Expansion of G_k We should argue that the final graph G_k in step 4, has every component to be an expander. Notice that the product operation happens to each component. Consider any component of the graph. If $\lambda_2(G_{k-1}) \leq 1 - \epsilon$, then $\lambda_2(G_{k-1} \circledR H) \leq 1 - \frac{\epsilon}{25}$. Thus, $\lambda_2(G_k) \leq \left(1 - \frac{\epsilon}{25}\right)^{50} \leq 1 - 2\epsilon$. Thus, the spectral gap ϵ improves to 2ϵ while going from G_{k-1} to G_k. Since there is a spectral gap of $\frac{1}{12n^2}$ already, this implies, we should choose k such that:

$$2^k \left(\frac{1}{12n^2}\right) > \frac{1}{2}$$

This explains the choice of $k = O(\log n)$ in step 4.

Logspace implementation:

Fooling the Conjunction using k-wise Independence

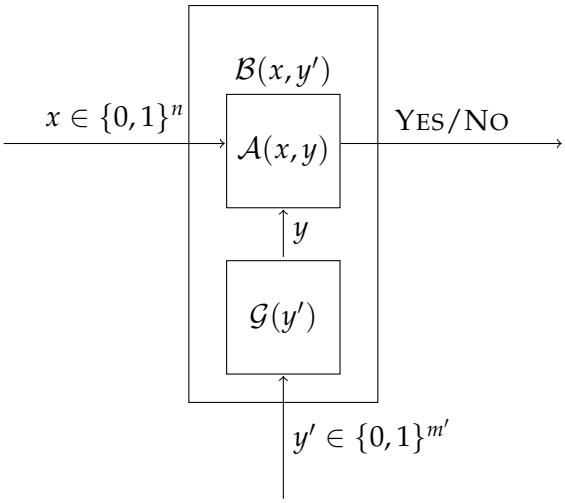
Lecturer : Jayalal Sarma

WEEK
10

We have seen limited independence in the previous lecture as a method for randomness efficient error reduction and sometimes complete derandomization of algorithms. Both of these can be thought of us means of studying limited independent bits $y \in \{0,1\}^m$ which can be generated by small number of bits as opposed to m independent random bits.

One of the views that we developed is that this can also be equivalently be viewed as, the limited independence is able to "fool" the amplification process. In this week, we will first explore this in a fundamental way by asking the following question - what kind of computation can limited independence fool? To formulate this further, let us recall the randomized algorithm set up once again.

Recall that we have a radomized algorithm $\mathcal{A}$ which takes in the input as $x \in \{0,1\}^n$ and random string is represented by $y \in \{0,1\}^m$. Our derandomization problem was equivalently stated as, given $\mathcal{A}(x)$ - with x fixed, as an algoithm which takes y as the input, with the guarantee that for most y's the algorithm outputs the correct answer $(0/1)$, can we algorithmically (efficiently) find out the correct answer given by the algorithm $\mathcal{A}$. For the purpose of understanding the idea of fooling restricted $\mathcal{A}$, we can imagine $\mathcal{A}$ as computing a Boolean function which transforms $y \in \{0,1\}^m$ bits to $\{0,1\}$. But notation, we will think of it as $f : \{0,1\}^m \to \{0,1\}$.



With the above view, we can define the follwing - a distribution D of m bits, $y \in \{0,1\}^m$, is said to be (s, ϵ)-pseudo-random if for every function f (which abstracts out the algorithm $\mathcal{A}$) which can be computed resource bounds[24] of s, we have that

$$\left| \Pr_{Y \leftarrow U}[f(Y) = 1] - \Pr_{Y \leftarrow D}[f(Y) = 1] \right| \leq \epsilon$$

We start with the task of fooling very simple Boolean functions f. Indeed, the task is not interesting if f is not even depending on all input bits. Say if it depends only on the first bit y_1 of y, then we can fool the computation by just keeping y_1 as a pure random bit and the rest of the bits dependent

[24] We keep this term to be resource bounds when considering computation of Boolean functions. A natural model of computation here is that of circuits, which we describe later when it is needed.

on it. Indeed, arguably the first Boolean function that we can think of would be the conjuction and disjunction and another one would be parity function, majority function etc. We take them up in the next two sections.

10.1 Fooling the Conjuction ($\wedge_n$) using k-wise Independece

Now we show that the good old k-wise independent distributions that we have constructed in the earlier lectures actually fools the conjuction with an exponentially decaying ϵ with respect to k. Formally,

THEOREM 10.1.1. *Let D be a k-wise independent distribution on $\{0,1\}^m$. Then, there is a constant c such that :*

$$\left| \Pr_{Y \leftarrow U} \left[\bigwedge_{i=1}^{m} Y_i = 1 \right] - \Pr_{Y \leftarrow D} \left[\bigwedge_{i=1}^{m} Y_i = 1 \right] \right| \leq \frac{1}{2^{ck}}$$

Proof. The main technique in the proof is the inclusion exclusion principle and some clever approximations. The original published proof that we give here is due to

To start with, we will translate the statement about intersection to unions by applying De-Morgan's laws.

The following discussion will hold true for any distribution D on $\{0,1\}^m$.

$$\begin{aligned} \Pr_{Y \leftarrow D} \left[\bigwedge_{i=1}^{m} Y_i = 1 \right] &= 1 - \Pr_{Y \leftarrow D} \left[\bigwedge_{i=1}^{m} Y_i = 0 \right] \\ &= 1 - \Pr_{Y \leftarrow D} \left[\bigcup_{i=1}^{m} E_i \right] \text{ where } E_i \text{ is the event } Y_i = 0. \end{aligned}$$

Using this, we can rewrite what we want to estimate:

$$\left| \Pr_{Y \leftarrow U} \left[\bigwedge_{i=1}^{m} Y_i = 1 \right] - \Pr_{Y \leftarrow D} \left[\bigwedge_{i=1}^{m} Y_i = 1 \right] \right| = \left| \Pr_{Y \leftarrow U} \left[\bigcup_{i=1}^{m} E_i \right] - \Pr_{Y \leftarrow D} \left[\bigcup_{i=1}^{m} E_i \right] \right|$$

Thus, our task reduces to estimating $\Pr_{Y \leftarrow D} \left[\bigcup_{i=1}^{m} E_i \right]$. This s where naturally the principle of exclusion can be applied. Recalling the principle : the starting point is the union bound - $\Pr \left[\bigcup_{i=1}^{m} E_i \right] \leq \Pr[E_1] + \Pr[E_2] + \ldots + \Pr[E_m]$ which forms an upper bound. In an attempt to achieve equality, we subtract the double-counted terms, and this leads to an over-subtraction, which we add again and so on. More formally, we write it in terms of the index sets:

$$\Pr\left[\bigcup_{i=1}^{m} E_i\right] \leq \sum_{\substack{S\subseteq[m]\\|S|=1}} \Pr\left[\bigcap_{i\in S} E_i\right]$$

$$\Pr\left[\bigcup_{i=1}^{m} E_i\right] \geq \sum_{\substack{S\subseteq[m]\\|S|=1}} \Pr\left[\bigcap_{i\in S} E_i\right] - \sum_{\substack{S\subseteq[m]\\|S|=2}} \Pr\left[\bigcap_{i\in S} E_i\right]$$

$$\Pr\left[\bigcup_{i=1}^{m} E_i\right] \geq \sum_{\substack{S\subseteq[m]\\|S|=1}} \Pr\left[\bigcap_{i\in S} E_i\right] - \sum_{\substack{S\subseteq[m]\\|S|=2}} \Pr\left[\bigcap_{i\in S} E_i\right] + \sum_{\substack{S\subseteq[m]\\|S|=3}} \Pr\left[\bigcap_{i\in S} E_i\right]$$

And in general - if we define : $T_i = \sum_{\substack{S\subseteq[m]\\|S|=i}} \Pr\left[\bigcap_{j\in S} E_j\right]$ and the partial sums $S_\ell = \sum_{i=1}^{\ell}(-1)^{i+1}T_i$. Using this, we conclude:

$$\forall 1 \leq \ell \leq m, \quad \Pr\left[\bigcup_{i=1}^{m} E_i\right] \quad \text{is} \quad \begin{cases} \geq S_\ell \text{ when } \ell \text{ is odd.} \\ \leq S_\ell \text{ when } \ell \text{ is even.} \end{cases}$$

Whatever we discussed so far is true for any distribution D. Let us create notation for the two quantities that we want to estimate:

$$\Gamma = \Pr_{Y\leftarrow D}\left[\bigcup_{i=1}^{m} E_i\right] \qquad\qquad \Delta = \Pr_{Y\leftarrow D}\left[\bigcup_{i=1}^{m} E_i\right]$$

and we want to estimate an upper bound for $|\Gamma - \Delta|$.

Now we apply the fact that the distribution D is k-wise independent. As per the definition, for any subset $S \subset [m]$,

$$\Pr_{D\leftarrow Y}\left[\bigcap_{i\in S} E_i = 0\right] = \prod_{i\in S} \Pr_{D\leftarrow Y}[E_i]$$

and this is true for both the k-wise independent distribution D and the uniform distribution U (which is n-wise independent. Interpreting in terms of the above notation, this implies that $\forall \ell \leq k$, the value of S_i remains the same whether we work with the distribution D or U.

This in particular, implies that :

$$\forall \ell : 1 \leq \ell \leq k-1, \text{ where } \ell \text{ is even, } S_\ell \leq \Gamma \leq S_{\ell+1} \text{ and } S_\ell \leq \Delta \leq S_{\ell+1}$$

This gives that

$$|\Gamma - \Delta| \leq |S_{\ell+1} - S_\ell| \leq T_\ell$$

Noticing that the gap between S_ℓ and $S_{\ell+1}$ keeps on decreasing with increasing ℓ, we have that $|\Gamma - \Delta| \leq T_k$. Now we need to estimate an upper bound for T_k. □

Estimating an upper bound for T_k : Now we estimate the value of T_ℓ which involves a useful generalization of the arithemetic mean - geometric mean inequality.

LEMMA 10.1.2. *Let $q_1, q_2, \ldots q_m$ be non-negative real numbers. If we choose $S \subset [m]$:*

$$\mathbb{E}_{\substack{S\subseteq[m]\\|S|=1}}\left[\prod_{j\in S} q_j\right] \geq \mathbb{E}_{\substack{S\subseteq[m]\\|S|=2}}\left[\left(\prod_{j\in S} q_j\right)^{1/2}\right] \geq \ldots \leq \mathbb{E}_{\substack{S\subseteq[m]\\|S|=k}}\left[\left(\prod_{j\in S} q_j\right)^{1/k}\right] \geq \ldots \geq \mathbb{E}_{\substack{S\subseteq[m]\\|S|=m}}\left[\left(\prod_{j\in S} q_j\right)^{1/m}\right]$$

The first term on the LHS is nothing but $\frac{1}{m}\sum_{i=1}^{m} q_i$ and the last term in the rightmost end is $\left(\prod_{j\in S} q_j\right)^{1/m}$ and hence the above is a generalization of AM-GM inequality (Exercise : Prove it precisely).

To apply this the lemma to bound T_k, we set up: (denote the $P_i = \Pr[E_i]$)

$$T_k = \sum_{\substack{S\subseteq[m]\\|S|=k}} \Pr_{Y\leftarrow D}\left[\bigcap_{j\in S} E_j\right] = \sum_{\substack{S\subseteq[m]\\|S|=k}}\left[\prod_{i\in S} P_i\right] \leq \binom{n}{k}\mathbb{E}_{\substack{S\subseteq[m]\\|S|=k}}\left[\prod_{i\in S} P_i\right]$$

Now we can apply the Lemma to the RHS expression, and get the following :

$$T_k \leq \binom{n}{k}\sum_{i=1}^{m}\left(\frac{1}{m}\sum_{i=1}^{m} P_i\right)^k \leq \left(\frac{em}{k}\right)^k\left(\frac{\sum P_i}{m}\right)^k = \left(\frac{e\sum P_i}{k}\right)^k$$

Now we have an easy case. Suppose $\sum P_i \leq \frac{2e}{k}$, then we are done. That is, if the individual bits are 0 with very low probability, then we are done. But this is hardly an interesting case for us.

It remains to handle the case when $\sum_i P_i > \frac{k}{2e}$. Let m' be the largest such that $\sum_{i=1}^{m'} P_i \leq \frac{k}{2e}$. We can apply the previous argument for the first m' bits.

Now let us define the following quantities:

$$V_D = \Pr_{Y\leftarrow D}\left[\bigwedge_{i=1}^{m} Y_i = 1\right] \qquad V_U = \Pr_{Y\leftarrow U}\left[\bigwedge_{i=1}^{m} Y_i = 1\right]$$

$$V'_D = \Pr_{Y\leftarrow D}\left[\bigwedge_{i=1}^{m'} Y_i = 1\right] \qquad V'_U = \Pr_{Y\leftarrow U}\left[\bigwedge_{i=1}^{m'} Y_i = 1\right]$$

What we need to upper bound is the expression $|V_D - V_U|$. By definition, $V_U \leq V'_U$, $V_D \leq V'_D$ since the event $\bigwedge_{i=1}^{m} Y_i = 1$ implies $\bigwedge_{i=1}^{m'} Y_i = 1$. Due to the choice of m', by applying the above easy case, we have that $|V'_D - V'_U| \leq 2^{-k}$.

Thus it suffices to prove that $V'_U \leq 2^{-\Omega(k)}$. We do this below as the last step of the proof.

$$
\begin{aligned}
V'_U &= \Pr_{Y \leftarrow U}\left[\bigwedge_{i=1}^{m'} Y_i = 1\right] = \prod_{i=1}^{n'} \Pr_{Y \leftarrow U}[Y_i = 1] = \prod_{i=1}^{n'}(1 - P_i) \\
&\leq \left(\frac{1}{m'}\sum_{i=1}^{m'}(1 - P_i)\right)^{m'} \text{ (by applying AM-GM inequality)} \\
&\leq \left(\frac{1}{m'}\left(m' - \sum_{i=1}^{m'} P_i\right)\right)^{m'} \leq \left(\frac{1}{m'}\left(m' - \frac{k}{2e}\right)\right)^{m'} \text{ (since } \sum P_i > \frac{k}{2e}\text{)} \\
&\leq \left(1 - \frac{k}{2em'}\right)^{m'} \leq e^{-\frac{k}{2e}} \text{ by definition of } e^x \\
&\leq e^{-\Omega(k)} \leq 2^{-\Omega(k)}
\end{aligned}
$$

Exercise 10.1.3 (See Problem Set 4 (Problem 1))**.** This question discusses a construction of an explicit $\frac{1}{2}$-hitting disperser from an explicit pairwise independent hash family. Given parameters $V = \{1, 2, \ldots m\}$ and $\epsilon > 0$, let D be a set of size more than $\lceil \frac{1}{\epsilon} \rceil$. Let H be a pairwise independent hash family that has functions from D to V such that the size of H is polynomial in $|D|/|V|$. Define the following disperser (U, V, E), with $U = H$ and let

$$E = \{(h, h(x)) \mid h \in U, x \in D\}$$

That is, each vertex of U corresponds to a hash function, and each outgoing edge from such a vertex corresponds to applying this hash function to a particular value in D. Show that what we constructed is a $\frac{1}{2}$-hitting disperser with $|U|$ polynomially bounded in $|V|/\epsilon$ with degree $O(\frac{1}{\epsilon})$ and threshold $\epsilon|U|$.

Hint : It suffices to show that if we take any subset V' of V of size at least $\frac{|V|}{2}$ and a random vertex $h \in U$, the probability that there is an edge between h and V' is more than $1 - \epsilon$. Show that this sufficient and then prove the same. In the probability calculation, Chebychev inequality and part (c) of a question from midsem will come handy, which you can use without proof for the purpose of this question.

Fooling the PARITY ($\oplus_n$) : Small Biased Sets

Lecturer : Jayalal Sarma

WEEK

11

Now we come to fooling PARITY of m bits. We start with the following notion of "almost uniform" distributions. Consider the following claim that we have seen before. For a $w \in \{0,1\}^m$, where $w \neq 0$, the probability that a randomly chosen $a \in \{0,1\}^m$ satisfies $\langle w, a \rangle = 0$ is exactly $\frac{1}{2}$. How close is this probability to half can be used as a measure of how pure the m bits are. The following definition formalizes this.

DEFINITION 11.0.4 (**Small Biased Distribution**). *Let $\epsilon > 0$. A distribution $Y = (Y_1, Y_2, \ldots, Y_m)$ over $\{0,1\}^m$ is said to be an ϵ-biased distribution if $\forall w \in \{0,1\}^m$:*

$$\frac{1-\epsilon}{2} \leq \Pr_{y \sim Y}[\langle w, y \rangle = 0] \leq \frac{1+\epsilon}{2}$$

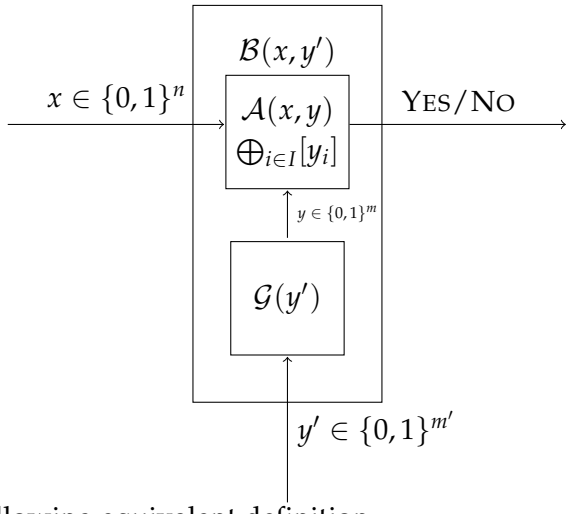
An equivalent definition is in terms of the psuedorandom generators that we discussed in the initial lectures. The algorithm $\mathcal{A}$ that we attempt to "fool" is rather simplistic, it is just the parity of a set of y bits (specified by the subset $I \subseteq [k]$).

By our formal definition of "fooling", this is exactly, for any $I \subseteq [m]$:

$$\left| \Pr_{y \sim D} \left(\bigoplus_{i \in I} y_i = 0 \right) - \Pr_{y \sim U} \left(\bigoplus_{i \in I} y_i = 0 \right) \right| \leq \frac{\epsilon}{2}$$

Noting that the second term is exactly $\frac{1}{2}$, this is equivalent definition to : for $I \subseteq [m]$,

$$\frac{1-\epsilon}{2} \leq \Pr_{y \sim D} \left(\bigoplus_{i \in I} y_i = 0 \right) \leq \frac{1+\epsilon}{2}$$



The small biased distributions get their name by the following equivalent definition.

DEFINITION 11.0.5 (**Bias of a distribution**). *A distribution $Y \subseteq \{0,1\}^m$ is said to have a bias of ϵ if for any $I \subseteq [m]$:*

$$\left| \Pr_{y \sim D} \left(\bigoplus_{i \in I} y_i = 0 \right) - \Pr_{y \sim D} \left(\bigoplus_{i \in I} y_i = 1 \right) \right| \leq \epsilon$$

It follows from the above that ϵ-biased distributions is exactly same as distributions whose bias is ϵ. The following characterization is also useful. Let D be a distribution over $\{0,1\}^m$.

PROPOSITION 11.0.6. *D is ϵ-biased* $\iff \forall w \in \{0,1\}^m \setminus \{0^m\}$, $\mathbb{E}_{y \to D}\left[(-1)^{\langle w,y\rangle}\right] \leq \epsilon$.

In the discussion below, for showing that some distributions are ϵ-biased, we show the RHS.

Small Biased Sets: For a distribution Y, the support of the distribution $\mathsf{supp}(Y)$ are the elements of the sample space which has a non-zero probability assigned to them. For the ϵ-biased distributions, we would like smaller support. Moreover, a special case is when we have a multiset of small size over which the distribution is uniform. This motivates the following definition.

DEFINITION 11.0.7 **(Small Biased Sets)**. *Let $\epsilon > 0$. A sub(multi)set $S \subseteq \{0,1\}^m$ is said to be an ϵ-biased set if the distribution Y on $\{0,1\}^m$ defined as:*

$$Y(w) = \begin{cases} 1/|S| & \text{if } w \in S \\ 0 & \text{otherwise} \end{cases}$$

is an ϵ-biased distribution on $\{0,1\}^m$.

The size of small biased set is an important parameter. Imagine that we have $S \subseteq \{0,1\}^m$ such that $|S| = \mathsf{poly}(m)$ to be small biased set, and that the set S is explicitly described an indexed by $\alpha \in \{0,1\}^\ell$ where $\ell = \log(|S|)$. Then, by choosing $O(\log m)$ bits uniformly at random, we have an ϵ-biased distribution which in certain situations will be as good as uniform distribution on $\{0,1\}^m$ (which requires m bits).

We will quickly remark that ϵ-biased distributions if efficiently and explcitly constructed will lead to newer constructions of expanders and newer t-wise independent distributions. We will present these later in this lecture.
We quickly remark on what is known:

- By probablistic method, we can show that there exists ϵ-biased spaces of size $O(m/\epsilon^2)$.
- The first explicit construction was by [Naor and Naor, 1990, Naor and Naor, 1993]. The space was of size $O(m/\epsilon^3)$.
- Incomparable bounds by [Alon et al., 1992]. The space constructed is of size $O(m^2/\epsilon^2)$.
- Improved bounds by [Ben-Aroya and Ta-Shma, 2009]. The space constructed is of size $O\left((m/\epsilon^2)^{5/4}\right)$.
- Almost optimal bounds by [Ta-Shma, 2017]. The space constructed is of size $O\left(\frac{m}{\epsilon^{2+o(1)}}\right)$.

While it may look like a hard fight for optimizing the power for ϵ in the above expression, there are applications where that decides the boundary of efficiency.

11.1 Expanders from Small Biased Sets

We now describe a connection between small biased sets and expanders. Suppose that $S \subseteq \mathbb{F}_2^m$ is an ϵ-biased set. We claim that this set naturally defines an expander. Viewing $\mathbb{F}_2^m$ as a group, a standard graph associated with it is the Cayley graph, defined as follows:

DEFINITION 11.1.1 (**Cayley Graph of $\mathbb{F}_2^m$ with respect to S**). *For $S \subseteq \mathbb{F}_2^n$, let $G(V, E)$ be the graph defined as follows. $V = \mathbb{F}_2^n$. Define the edges as:*

$$E = \{(a, a + w) \mid a \in \mathbb{F}_2^m \text{ and } w \in S\}$$

The graph is undirected since w is its own additive inverse. The number of vertices is 2^m, the degree is exactly $|S|$, assuming multiedges where elements repeat. We prove the following lemma.

LEMMA 11.1.2. *The graph G is $(2^m, |S|, \epsilon)$ spectral expander.*

Proof. Let A be the normalized adjacency matrix which is of the order $2^m \times 2^m$. We explicitly write down all the linearly independent eigen vectors and then bound the second largest eigen value.

For $y \in \mathbb{F}_2^n$, define a function $\Gamma_y : \mathbb{F}_2^n \to \mathbb{R}$ as :

$$\forall w \in \mathbb{F}_2^n \text{ define } \Gamma_y(w) = (-1)^{\langle y, w \rangle}$$

This function has some nice properties. For any $y \in \mathbb{F}_2^m$, $\Gamma_y(w + w') = \Gamma_y(w)\Gamma_y(w')$. Note that, given a $y \in \mathbb{F}_2^m$, we can consider Γ_y as a vector in $\mathbb{R}^{2^m}$. We claim:

CLAIM 11.1.3. *Γ_y are eigen vectors of A.*

Proof. We prove this directly by checking what $A\Gamma_y$ vector will be. Indeed:

$$\begin{aligned}
\forall a \in \{0,1\}^m : \quad \left(A\Gamma_y\right)[a] &= \sum_{(a,b)\in E} (A_{ab}) \left(\Gamma_y\right)[b] \\
&= \frac{1}{|S|} \sum_{(a,b)\in E} \left(\Gamma_y\right)[b] = \frac{1}{|S|} \sum_{(a,b)\in E} \Gamma_y(b) \\
&= \frac{1}{|S|} \sum_{w \in S} \Gamma_y(a + w) = \Gamma_y(a) \left(\sum_{w \in S} \frac{1}{|S|} \Gamma_y(w) \right) \\
&= \lambda_y(\Gamma_y[a]) \\
A\Gamma_y &= \lambda_y \Gamma_y
\end{aligned}$$

And hence Γ_y is an eigen vector of A with eigen value $\sum_{w \in S} \frac{1}{|S|} \Gamma_y(w)$. ∎

Now we turn to bounding the eigen values λ_y. Notice that $\lambda_{0^m} = 1$. Indeed, when $y = 0$, the vector $\Gamma_y \in \mathbb{R}^{2^m}$ is the all 1s vector and hence is an eigen vector for the eigen value 1. We claim that when $y \neq 0$, $\lambda_y \leq \epsilon$. Let D be the distribution on $\{0,1\}^m$ with support as S and uniformly distributed over S. Since the set S is ϵ-biased, the distribution D has bias at most ϵ.

$$\begin{aligned}
\lambda_y = \sum_{w \in S} \frac{1}{|S|} \Gamma_y(w) &= \frac{1}{|S|} \sum_{w \in S} (-1)^{\langle y, w \rangle} = \frac{1}{|S|} \left(\sum_{\substack{w \in S \\ \langle y, w \rangle = 0}} (1) \right) + \frac{1}{|S|} \left(\sum_{\substack{w \in S \\ \langle y, w \rangle = 1}} (-1) \right) \\
&= \Pr_{w \sim D} [\langle y, w \rangle = 0] - \Pr_{w \sim D} [\langle y, w \rangle = 1] \leq \epsilon \qquad \text{since bias of } D \text{ is at most } \epsilon.
\end{aligned}$$

□

11.2 Existence of ϵ-Biased Sets

We now quickly show the existence of ϵ biased set of size $\frac{m}{\epsilon^2}$. The argument is through probabilistic method. We state the technical theorem first.

THEOREM 11.2.1. *For every ϵ, there exists $S \subseteq \{0,1\}^m$ of size $O(\frac{m}{\epsilon^2})$ such that S is an ϵ-biased set.*

Proof. We choose $z_1, z_2 \dots z_\ell$ uniformly at random from the set $S \subseteq \{0,1\}^m$. We imagine that $S = \{z_1, z_2, \dots z_\ell\}$. We ask the question. What does it mean for S to be ϵ biased? By definition, if Y is the distribution over $\{0,1\}^m$ with support as S (and distributed uniformly), we need that: $\forall w \in \{0,1\}^n \setminus \{0^m\}$:

$$\frac{1-\epsilon}{2} \leq \Pr_{y \sim Y}[\langle w, y \rangle = 0] \leq \frac{1+\epsilon}{2}$$

$$\text{Equivalently, } \frac{(1-\epsilon)\ell}{2} \leq \left|\{i \in [\ell] \mid \langle w, z_i \rangle = 0\}\right| \leq \frac{(1+\epsilon)\ell}{2}$$

$$\text{Equivalently, } \left|\left|\{i \in [\ell] \mid \langle w, z_i \rangle = 0\}\right| - \frac{\ell}{2}\right| \leq \frac{\epsilon\ell}{2}$$

To show the existence of the set S of the required size, we need to show that for ℓ chosen as required, we should prove :

$$\Pr_{z_1, z_2, \dots z_\ell}\left[\begin{array}{c} \forall w \in \{0,1\}^m, w \neq 0 \\ \left|\left|\{i \in [\ell] \mid \langle w, z_i \rangle = 0\}\right| - \frac{\ell}{2}\right| \leq \frac{\epsilon\ell}{2} \end{array}\right] > 0$$

It suffices to show an upper bound on the complementary event.

$$\Pr_{z_1, z_2, \dots z_\ell}\left[\begin{array}{c} \exists w \in \{0,1\}^m, w \neq 0 \\ \left|\left|\{i \in [\ell] \mid \langle w, z_i \rangle = 0\}\right| - \frac{\ell}{2}\right| > \frac{\epsilon\ell}{2} \end{array}\right] < 1$$

We plan to apply union bound to handle $\exists w \in \{0,1\}^m$ part of the statement. Hence, we fix a $w \in \{0,1\}^m$, such that $w \neq 0$ and want to derive an upper bound for:

$$\Pr_{z_1, z_2, \dots z_\ell}\left[\left|\left|\{i \in [\ell] \mid \langle w, z_i \rangle = 0\}\right| - \frac{\ell}{2}\right| > \frac{\epsilon\ell}{2}\right]$$

To model this, define a random variable X_i which takes value 1 when z_i satisfies $\langle w, z_i \rangle = 0$ and 0 otherwise. Since z_i is chosen uniformly at random and $w \neq 0$, $\mathbb{E}[X_i] = \frac{1}{2}$. Defining $X = \sum_{i=1}^{\ell}$ gives us $\mathbb{E}[X] = \frac{\ell}{2}$. We are asking for the probability that $\Pr_{z_1, z_2, \dots z_\ell}\left[\left|X - \mathbb{E}[X]\right| \geq \frac{\epsilon\ell}{2}\right]$. By Chernoff's bound[25]

$$\Pr_{z_1, z_2, \dots z_\ell}\left[\left|X - \mathbb{E}[X]\right| \geq \frac{\epsilon\ell}{2}\right] \leq 2^{-\Omega\left(\frac{\epsilon^2\ell}{2}\right)}$$

[25] If $X = \sum_{i=1}^{n} X_i$ then $\Pr\left[\left|X - \mathbb{E}[X]\right| \geq A\right] \leq e^{-A^2/2n}$

With the union bound applied:

$$\Pr_{z_1,z_2,\ldots z_\ell}\left[\begin{array}{c}\exists w \in \{0,1\}^m, w \neq 0 \\ \left|\left|\{i \in [\ell] \mid \langle w, z_i\rangle = 0\}\right| - \frac{\ell}{2}\right| > \frac{\epsilon\ell}{2}\end{array}\right] \leq 2^m \times 2^{-\Omega\left(\frac{\epsilon^2\ell}{2}\right)}$$

For this to be less than 1, we just need to choose ℓ such that $m < \frac{\epsilon^2\ell}{2}$. Thus, choice of $\ell = O\left(\frac{m}{\epsilon^2}\right)$ works. Hence, by probabilistic method, there exists a set of size $O\left(\frac{m}{\epsilon^2}\right()$ which is an ϵ-biased set. This completes the proof of existence. □

11.3 Explicit Construction of ϵ-biased Sets

We first digress a little bit and understand the connection between two finite field $\mathbb{F}_q^m$ and $\mathbb{F}_{q^m}$. Let $r(x)$ be an irreducible polynomial[26] of degree m in $\mathbb{F}_q[x]$ - which are polynomials whose coefficients are from $\mathbb{F}_q$. We claim that the set:

$$\mathbb{F} \stackrel{\text{def}}{=} \{p(x) \mod r(x) : p(x) \in \mathbb{F}_q[x]\}$$

forms a finite field where the addition and multiplication is modulo the polynomial $r(x)$. By notation, $\mathbb{F} \equiv \mathbb{F}/\langle f\rangle$. Notice that, since the set is finite and there is no zero divisor in the set ($a, b \neq 0$ such that $ab = 0$), every non-zero element in $\mathbb{F}$ has an inverse[27]. Hence this set is a field. Thus, each distinct element in $\mathbb{F}$ can also be viewed as a tuple of coefficients $(c_0, c_1, c_2, \ldots c_{m-1}) \in (\mathbb{F}_q)^m$ and vice versa. Thus it is a vector space over $\mathbb{F}_q$ of dimension m.

For the remaining discussion, $r(x) = a_0 + a_1x + \ldots a_mx^m$. Let α be a root of $r(x)$. Clearly, since $r(x)$ is irreducible, α is not in $\mathbb{F}_q$ (hence, consider α as an abstract symbol for a root of $r(d)$). Let us "adjoin" α to $\mathbb{F}_q$ and then try to take closure to make it a field. To start with, what is α^2?, that becomes abstract another element outside $\mathbb{F}_q$, this way, we introduce $m-1$ new elements, $\alpha, \alpha^2, \ldots \alpha^{m-1}$. At the m^{th} step, what is α^m? That is not a new element since $r(\alpha) = 0$ and $a_m \neq 0$, hence $\alpha^m = \frac{1}{a_n}(-a_0 - a_1\alpha - a_2\alpha^2 - \ldots - a_{n-1}\alpha^{n-1})$. But then, all the distinct combinations of the above powers of α with coefficients from $\mathbb{F}_q$ are all distinct elements since α does not introduce any non-trivial relationship among them. This gives rise to a set of size q^m which is a field by construction (again, inverse exists because it is a finite ring without zero divisors). We will call this set to be $\mathbb{F}_{q^m}$.

The two sets that we described above $\mathbb{F}_{q^m}$ and $(\mathbb{F}_q)^m$ have a natural bijection among them. The map from $(\mathbb{F}_q)^m$ to $\mathbb{F}_{q^m}$ is easy to describe : given $(c_0, c_1, c_2, \ldots c_{m-1}) \in (\mathbb{F}_q)^m$, which denotes the polynomial $p(x) = c_0 + c_1x + c_2x^2 + \ldots c_{m-1}x^{m-1}$. The image in $\mathbb{F}_{q^m}$ is the evaluation $p(\alpha)$. This map is a bijection and also respects the addition and multiplication. It is an isomorphism between the two fields. Thus, we have an isomorphism, $\phi : \mathbb{F}_{q^m} \to \mathbb{F}_q^m$. We will use this map for the construction.

[26] A polynomial is said to be irreducible if it cannot be written as the product of two other polynomials of smaller degree.

[27] Consider $a \neq 0$, and suppose a does not have an inverse, then mutliplying a with all the non-zero elements in $\mathbb{F}$ should not give repeated elements (otherwise it will give zero divisors) and should not contain 0 and 1. Hence a contradiction.

Construction of the ϵ-biased set: We directly define the set:

$$S = \left\{ y \in \{0,1\}^m \;\middle|\; \begin{array}{c} y = y_1 y_2 \ldots y_m \text{ where} \\ y_i = \langle \phi(\alpha^i), z \rangle, \text{ for } \alpha, z \in \mathbb{F}_2^\ell \end{array} \right\}$$

As per the above formulation, the size of the set S is at most $2^{2\ell}$. We will choose the parameter ℓ later (to be $\log\left(\frac{m}{\epsilon}\right)$) and this will meet our size requirement. We now prove that for the above value of ℓ, the set S is ϵ-biased.

LEMMA 11.3.1. *For $\ell = \log\left(\frac{m}{\epsilon}\right)$), the set S is ϵ-biased.*

Proof. Consider the distribution Y on $\{0,1\}^m$ defined as:

$$Y(y) = \begin{cases} 1/|S| & \text{if } y \in S \\ 0 & \text{otherwise} \end{cases}$$

For a given $w \in \{0,1\}^m$, $w \neq 0$, we need to estimate the probability $\Pr_{y \sim Y}[\langle y, w \rangle = 0]$. This is same as:

$$\begin{aligned} Pr_{y \in S}[\langle y, w \rangle = 0] &= \Pr_{y \in S}\left(\sum_{i=1}^m y_i w_i = 0 \right) = \Pr_{y \in S}\left[\sum_{i=1}^m \langle \phi(\alpha^i), z \rangle w_i = 0 \right] \\ &= \Pr_{y \in S}\left[\sum_{i=1}^m \langle \phi(\alpha^i) w_i, z \rangle = 0 \right] = \Pr_{y \in S}\left[\sum_{i=1}^m \langle \phi(\alpha^i w_i), z \rangle = 0 \right] \\ &= \Pr_{y \in S}\left[\left\langle \sum_{i=1}^m \phi(\alpha^i w_i), z \right\rangle = 0 \right] = \Pr_{y \in S}\left[\left\langle \phi\left(\sum_{i=1}^m \alpha^i w_i \right), z \right\rangle = 0 \right] \\ &= \Pr_{y \in S}[\langle \phi(p_w(\alpha)), z \rangle = 0] \end{aligned}$$

where $p_w(x) = \sum_{i=1}^m w_i x^i$ is a polynomial defined by w of degree m.

Notice that choosing $y \in S$ unformly at random is equivalent to choosing α and z uniformly at random from $\mathbb{F}_2^\ell$. Since ϕ takes only 0 to 0^m, $\phi(p_w(\alpha)) = 0$ if and only if $p_w(\alpha) = 0$. Hence, we can estimate the above expression by conditioning on the event $p_w(\alpha) = 0$.

$$\begin{aligned}\Pr_{\alpha,z\in\mathbb{F}_2^\ell}[\langle\phi(p_w(\alpha)),z\rangle=0] &= \Pr_{\alpha,z\in\mathbb{F}_2^\ell}[\langle\phi(p_w(\alpha)),z\rangle=0 \mid p_w(\alpha)=0]\Pr_{\alpha\in\mathbb{F}_2^\ell}[p_w(\alpha)=0]\\ &\quad+\Pr_{\alpha,z\in\mathbb{F}_2^\ell}[\langle\phi(p_w(\alpha)),z\rangle=0 \mid p_w(\alpha)\neq 0]\Pr_{\alpha\in\mathbb{F}_2^\ell}[p_w(\alpha)\neq 0]\end{aligned}$$

$\Pr_{\alpha\in\mathbb{F}_2^\ell}[p_w(\alpha)=0]\leq\frac{m}{2^\ell}$ since $p_w(x)$ can have at most m roots in $\mathbb{F}_2^\ell$. Denote $p=\frac{m}{2^\ell}$. Conditioned on $p_w(\alpha)=0$, we know that $\phi(p_w(\alpha))=0$ and, $\Pr_{\alpha,z\in\mathbb{F}_2^\ell}\langle\phi(p_w(\alpha)),z\rangle=0]=1$. Conditioned on $p_w(\alpha)\neq 0$, we know that $\phi(p_w(\alpha))\neq 0$ and, $\Pr_{\alpha,z\in\mathbb{F}_2^\ell}\langle\phi(p_w(\alpha)),z\rangle=0]=\frac{1}{2}$. Thus,

$$\Pr_{\alpha,z\in\mathbb{F}_2^\ell}[\langle\phi(p_w(\alpha)),z\rangle=0]\leq 1\times p+\frac{1}{2}\times(1)\leq\frac{1}{2}+p$$

To see a lower bound,

$$\begin{aligned}\Pr_{\alpha,z\in\mathbb{F}_2^\ell}[\langle\phi(p_w(\alpha)),z\rangle=0] &\geq \Pr_{\alpha,z\in\mathbb{F}_2^\ell}[\langle\phi(p_w(\alpha)),z\rangle=0 \mid p_w(\alpha)\neq 0]\Pr_{\alpha\in\mathbb{F}_2^\ell}[p_w(\alpha)\neq 0]\\ &\geq \frac{1}{2}(1-p)\geq\frac{1}{2}-\frac{p}{2}\geq\frac{1}{2}-p\end{aligned}$$

Hence,

$$\text{Hence, }\frac{1}{2}-p\leq\Pr_{y\sim Y}[\langle y,w\rangle=0]\leq\frac{1}{2}+p$$

Thus, we just need to ensure that, $p\leq\frac{\epsilon}{2}$. We choose ℓ such that, $2^\ell=\frac{m}{2\epsilon}$. Recall that the set S was of size $2^{2\ell}$. Hence $|S|=\frac{m^2}{4\epsilon^2}\leq O\left(\frac{m^2}{\epsilon^2}\right)$. This completes the correctness proof of the construction. □

11.4 Repeated Sampling to Improves Bias

Consider an ϵ-biased set $S\subseteq\{0,1\}^m$. As earlier, the distribution D defined based on the set S on $\{0,1\}^m$:

$$D(y)=\begin{cases}1/|S| & \text{if } y\in S\\ 0 & \text{otherwise}\end{cases}$$

By definition D is ϵ-biased. We will consider the following distribution derived from D. Let $y_1,y_2,\ldots y_t$ be t strings in in $\mathbb{F}_2^m$, independently chosen from distribution D. Consider the distribution of their sum $y=y_1+y_2+\ldots+y_t$ where the sum is defined over $\mathbb{F}_2^m$ (and hence is bitwise sum modulo 2). We will denote the distribution of y over $\{0,1\}^m$ to be D'. We claim that D' is ϵ^t biased.

CLAIM 11.4.1. *The distribution* D^t *is* ϵ^t*-biased.*

Proof. Let $w\in\{0,1\}^m$ such that $w\neq 0$. By Proposition 11.0.6 it suffices to upper bound

$\mathbb{E}_{y\sim D^t}(-1)^{\langle y,w\rangle}$.

$$\begin{aligned}\mathbb{E}_{y\sim D^t}(-1)^{\langle y,w\rangle} &= \mathbb{E}_{y_1,y_2\ldots y_t\sim D}\left[(-1)^{\langle \sum_i y_i,w\rangle}\right] \\ &= \mathbb{E}_{y_1,y_2\ldots y_t\sim D}\left[\prod_i\left((-1)^{\langle y_i,w\rangle}\right)\right] = \prod_i \mathbb{E}_{y_i\sim D}\left[\left((-1)^{\langle y_i,w\rangle}\right]\right) \leq \epsilon^t\end{aligned}$$

□

Curiosity 11.4.2. The above is a trivial bias amplification method and the analysis depends on independence of the sampling. One may ask the question, can we do more randomness efficient amplification of the bias?

Given our background with expanders, it is natural to consider a random walk on expanders. A simple case is consider the following length 2 walk lemma which is attributed to Rozenman and Wigderson (unpublished).

LEMMA 11.4.3. *Let D be an ϵ-biased distribution over $\{0,1\}^m$ and let G be an $(2^m, d, \lambda)$ spectral expander whose vertices are labeled by samples of D so that the number of vertices labeled $w \in \{0,1\}^m$ is proportional to $D(w)$. Let D' be the following distribution: Uniformly choose a random vertex y_1 and choose a random neighbor of y_2 to sample a random edge (y_1, y_2) of G and output $y = y_1 + y_2$. Then D' is $(\epsilon^2 + \lambda)$-biased and with support $O(d.\mathsf{supp}(D))$.*

Notice that ϵ^2 is smaller than ϵ and that amplifies the bias, and we lose it out a bit by additive λ. Thus if we choose a better spectral expander, we achieve amplification without much loss. One can also repeat this again and again, and achieve a good bias amplification for resulting distribution D_i which will be ϵ_i-biased and support size s_i such that $\epsilon_{i+1} = \epsilon_i^2 + \lambda_i$ and $s_{i+1} = O(d.s_i)$. Choosing $\lambda_i = \epsilon_i^2$ and $d_i = 1/\lambda_i^4$, this gives:

$$\epsilon_{i+1} = 2\epsilon_i^2 \text{ and } s_{i+1} = s_i/\epsilon_i^4$$

Instead of repeating this on different graphs, one can imagine [Ta-Shma, 2017] taking a walk for t steps: that is, choose y_1 from D and then take a walk on the s-wide replacement product reduces the bias almost optimally. If the labels obtained in the walk be $y_1, y_2, y_3, \ldots y_t$ then output $y = y_1 + y_2 + \ldots + y_t$. Let D^t be the resulting distribution whose bias we need to estimate. [Ta-Shma, 2017] analyses this as follows: let $w \in \{0,1\}^m$ and $w \neq 0$. We need to show that:

$$\Pr_{y\sim D^t}[\langle y, w\rangle = 1] = \Pr_{y\sim D^t}\left[\sum_{i=1}^t \langle y_i, w\rangle = 1\right] \in \left[\frac{1}{2} - \frac{\delta}{2}', \frac{1}{2} + \frac{\delta}{2}\right]$$

Thus, if we define a bad set B as follows:

$$B = \left\{u \in \{0,1\}^m \mid \sum_{i:w_i=1} u_i = 1\right\}$$

We need to estimate the probability that $\sum_{i=1}^t \langle y_i, w\rangle = 1$. This is same as the probability that an odd number of y_is sampled falls into the bad set B. Compare this with the expander walk

based error reduction method, where we wanted to analyse the probability that the majority of the samples fall into a bad set. It is surprising that a similar analysis works for the "parity" of the samples to be in B also. We refer the reader to [Ta-Shma, 2017] for modeling this interesting fact algebraically and a proof. See Section 3 of [Ta-Shma, 2017].

11.5 Lowerbounds for the size of ϵ-biased Sets

We showed the construction of ϵ-biased set of size $\left(\frac{m}{\epsilon}\right)^2$ and also showed the existence of ϵ-biased set of size at most $\frac{m}{\epsilon^2}$. We now show that the existence proof is tight. That is any ϵ-biased set has to be that large.

THEOREM 11.5.1. *If $S \subseteq \{0,1\}^m$ is ϵ-biased, then $|S| \geq \Omega\left(\frac{m}{\epsilon^2 \log(1/\epsilon)}\right)$.*

Proof. As earlier, the distribution D defined based on the set S on $\{0,1\}^m$:

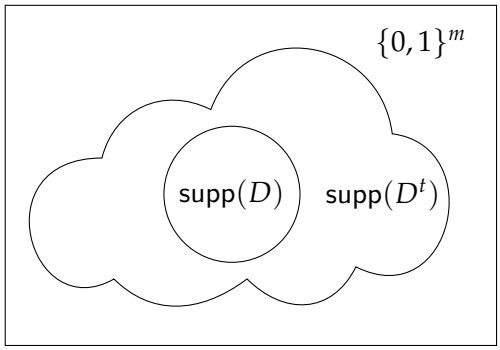
$$D(y) = \begin{cases} 1/|S| & \text{if } y \in S \\ 0 & \text{otherwise} \end{cases}$$

By definition D is ϵ-biased. We consider D^t which is ϵ^t-biased by Lemma 11.4.1.

Recall that the support of a distribution D ($\mathsf{supp}(D)$) is the elements of the sample space which has non-zero probability in the distribution. Eg: $\mathsf{supp}(D) = S$. We will study the size of support of D'. By definition:

$$\mathsf{supp}(D^t) = \left\{ y_1 + y_2 + \ldots + y_t : \begin{array}{c} \forall i \in [t] \\ y_i \in \mathsf{supp}(D) \end{array} \right\}$$

For picture assumes $0^m \in S$, and is shown for representational purposes.



We will estimate a lower bound and upper bound for the size of the $\mathsf{supp}(D)$ in terms of $|S|$.

Estimating an upper bound for $|\mathsf{supp}(D)|$: We would like to estimate

$$\left| \left\{ y_1 + y_2 + \ldots + y_t : \begin{array}{c} \forall i \in [t] \\ y_i \in \mathsf{supp}(D) \end{array} \right\} \right|$$

Let $s = |S| = |\mathsf{supp}(D)|$. Thus we need to choose $y_1, y_2, \ldots y_\ell \in S$ each of which repeats $c_1, c_2, \ldots, c_\ell$ such that $c_1 + c_2 + \ldots + c_\ell = t$. This is at most $\binom{s+t-1}{t} \leq \binom{s+t}{t} \leq \left(\frac{(s+t)e}{t}\right)^t$.

Estimating an lower bound for $|\mathsf{supp}(D)|$: We will use the following lemma which we leave as an exercise:

LEMMA 11.5.2. *If D is an ϵ-biased distribution, then as a vector $D \in \mathbb{R}^{2^m}$:*

$$\frac{1}{|\mathsf{supp}(D)|} \leq \|D\|_2^2 \leq \epsilon^2 + \frac{1}{2^m}$$

Applying this lemma to D^t which is ϵ^t biased: $\frac{1}{|\mathsf{supp}(D^t)|} \leq \left\|D^t\right\|_2^2 \leq \epsilon^{2t} + \frac{1}{2^m}$

Choose t such that $\epsilon^{2t} = 2^{-m}$, which is equivalent to $t = \frac{m}{2\log(1/\epsilon)}$. This gives,

$$\frac{1}{2\epsilon^{2t}} \leq |\mathsf{supp}(D^t)| \leq \left(\frac{(s+t)e}{t}\right)^t \text{ and hence, } \frac{1}{2^{1/t}\epsilon^2} \leq \frac{se}{t} + e$$

This gives : $|S| \geq \frac{m}{2^{1/t}} \left(\frac{1}{\epsilon^2 \log(1/\epsilon)}\right)$. Choose m large enought to get $|S| \geq \frac{m}{\epsilon^2 \log(1/\epsilon)}$. Hence the proof.

□

Exercise 11.5.3 (See Problem Set 4 (Problem 2))**.** If D is an ϵ-biased distribution over $\{0,1\}^m$, then as a vector $D \in \mathbb{R}^{2^m}$:

$$\frac{1}{|\mathsf{supp}(D)|} \leq \|D\|_2^2 \leq \epsilon^2 + \frac{1}{2^m}$$

Exercise 11.5.4 (See Problem Set 4 (Problem 3))**.** Let $G : \{0,1\}^{k \log m} \to \{0,1\}$ be the generator that we described such that $G\left(U_{k \log m}\right)$ outputs a k-wise independent distribution. (See the explicit construction of k-wise independent distribution). If we replace the input to G with a small bias distribution of $\epsilon' = \frac{\epsilon}{2^k}$, then the output of G is ϵ-close to being k-wise independent. Thus, conclude that, there is a generator for almost k-wise independent distributions with seed length $O\left(\log\log m + k + \log(1/\epsilon)\right)$.

11.6 ϵ-biased distributions are almost k-wise independent

In this lecture, we prove yet another connection between the mathematical objects that we have seen so far - namely, ϵ-biased sets and k-wise independent bits. Recall that in order to generate a distribution over $\{0,1\}^m$ which is k-wise indepdnent, we needed to invest $O(k \log n)$ random bits. In otherwise, the explicit k-wise independence construction that we saw earlier can also be seen as a function $G : \{0,1\}^{k \log m} \to \{0,1\}^m$ such that $G\left(U_{k \log m}\right)$ is a distribution that is k-wise independent. For generators, the number of pure random bits that they use, is called the *seed length*. So the seed length for our k-wise independent generator is $O(k \log m)$.

Suppose, we do not really require k-wise independent distributions, but are fine with "almost" k-wise independent distributions, the it turns out that ϵ biased sets can do the job with $O(k + \log m)$ seed instead of $O(k \log m)$. We define the notion of "almost" first. We say that the distribution is "almost" k-wise independent if every subset of k bits is "close" to uniform distribution. Now we need the definition of "closeness" of distribitions which we define first:

DEFINITION 11.6.1 (**δ-close Distributions**). *Two distributions D and D' over $\{0,1\}^m$ are said to be δ-close to each other, if for any two $T \subseteq \{0,1\}^m$, the probability that a randomly chosen string is in T is same for both distributions, upto additive δ. More formally:*

$$\forall T \subseteq \{0,1\}^m \ : \ \left| \Pr_{y \to D}\left[y \in T\right] - \Pr_{y \to D'}\left[y \in T\right] \right| \leq \delta$$

Note that, $T \subseteq \{0,1\}^m$ can equivalently be viewed as a Boolean function $T : \{0,1\}^m \to \{0,1\}$. We use the same notation for the subset and the function. In terms of this, the above condition can also be rewritten as:

$$\forall T \subseteq \{0,1\}^m \ : \ \left| \Pr_{y \to D}[T(y) = 1] - \Pr_{y \to D'}[T(y) = 1] \right| \leq \delta$$

Another way to view T is as a *distinguisher function* which tries to distinguish the two distributions. The statement of the definition says, no Boolean function test T will be able to distinguish the two distributions (in the sense that they do not have too different probabilities of getting 1 for randomly chosen t). with more than δ probability. Compare this with the definition of pseudorandom generators - where we have restrictions on T (which we called $\mathcal{A}$ in that context) based on how easy the function T is to compute.

Why do we care about almost k-wise independent distributions? This is a question we will not answer in detail as a good part of it is in the exercises. However, we will set up the background here.

Why are ϵ-biased distributions almost k-wise independent? We now prove the main theorem that we want to learn in this lecture. We will prove a more general theorem, which will imply the statement to k-wise independent distributions.

THEOREM 11.6.2. *If $\mathcal{D}$ is an ϵ-biased distribution over $\{0,1\}^m$, then it is δ-close to uniform distribution where $\delta = \epsilon 2^{m/2}$.*

To prove the above theorem, as in the definition of δ-closeness, we need to deal with arbitrary distinguisher Boolean functions T. For this, we use Fourier analysis as a tool:

Basic Fourier Analysis of Boolean Functions: Recall that we defined for $y \in \mathbb{F}_2^n$, the character function $\chi_y : \mathbb{F}_2^m \to \mathbb{R}$ as :

$$\forall w \in \mathbb{F}_2^m \text{ define } \chi_y(w) = (-1)^{\langle y,w \rangle}$$

This function has some nice properties. For any $y \in \mathbb{F}_2^m$, $\chi_y(w + w') = \chi_y(w)\chi_y(w')$. And, $\chi_y(w) = \chi_w(y)$. If we view $y \in \{0,1\}^m$ as the characterestic vector of a set $A \subseteq [m]$, then this can be interpreted as:

$$\forall w \in \{0,1\}^m, \text{ define } \chi_A(w) = (-1)^{\sum_{i \in A} w_i}$$

Note that, given a $A \subseteq [m]$, we can consider χ_A as a vector in $\mathbb{R}^{2^m}$. One interesting property is that these vectors (for different subsets $A \subseteq [m]$ are pairwise orthogonal (Prove this !). Hence they are also independent. Thus, they form a basis for $\mathbb{R}^{2^m}$ and is called the *Fourier basis*. Thus, any function $f : \{0,1\}^m \to \mathbb{R}$ (viewed as a vector in $\mathbb{R}^{2^m}$) is expressible as a linear combination of these vectors, as: $f = \sum_{A \subseteq [m]} \alpha_A \chi_A$. To standardise the notation, we denote the $\alpha_A \in \mathbb{R}$ as $\widehat{f}(A)$.

This gives the following expression for the evaluation of the function f on input s.

$$\forall w \in \{0,1\}^m : f(w) = \sum_{A \subseteq [m]} \widehat{f}(A)\chi_A(w)$$

This is called the *Fourier representation* of the function. The coefficients, $\widehat{f}(A)$ for different A are called the *Fourier coefficients* of the function f. Since the basis is orthogonal, we have the coefficients as : $\langle f, \chi_A \rangle$. Hence, for any $f : \{0,1\}^m \to \mathbb{R}$ we can define another function $\widehat{f} : \{0,1\}^m \to \mathbb{R}$ such that $\widehat{f}(A) = \langle f, \chi_A \rangle$. The linear transformation from $\mathbb{R}^{2^m}$ to $\mathbb{R}^{2^m}$ which transforms the vector f to $\widehat{f}$ is called the *Fourier transform*.

$$\widehat{f}(A) = \frac{1}{2^m} \sum_{w \in \{0,1\}^m} f(w)\chi_A(w) = \sum_{w \in \{0,1\}^m} \frac{1}{2^m} f(w)\chi_A(w) = \mathbb{E}_w \left[f(w)\chi_A(w) \right]$$

THEOREM 11.6.3 (**Plancherel Theorem**). *Let $f, g : \{0,1\}^m \to \mathbb{R}$. Then the following holds:*

$$\langle f, g \rangle = \sum_{S \subseteq [m]} \widehat{f}(S)\hat{g}(S)$$

Proof. We use the definitions:

$$\begin{aligned}
\langle f, g \rangle &= \left\langle \sum_{S \subseteq [m]} \widehat{f}(S)\chi_S, \sum_{T \subseteq [m]} \widehat{g}(T)\chi_T \right\rangle \\
&= \sum_{S \subseteq [m]} \widehat{f}(S) \left\langle \chi_S, \sum_{T \subseteq [m]} \widehat{g}(T)\chi_T \right\rangle \\
&= \sum_{S \subseteq [m]} \widehat{f}(S) \sum_{T \subseteq [m]} \widehat{g}(T) \langle \chi_S, \chi_T \rangle \\
&= \sum_{S \subseteq [m]} \widehat{f}(S)\widehat{g}(S) \qquad \because \chi_S, \chi_T \text{ are othognal when } S \neq T
\end{aligned}$$

Following is an important corollary, which can obtained by substututing $f = g$ in Plancharel's theorem and using the fact that the range of these functions are $\{-1, 1\}$.

COROLLARY 11.6.4 (**Parseval's Identity**). *If $f : \{0,1\}^m \to \{1, -1\}$ then :*

$$\sum_{T \subseteq [m]} \widehat{f}(S)^2 = 1$$

ϵ-biased distributions are δ-close to uniform distribution: We now come back to the proof of the theorem. We need to prove that for any distinguisher T, the difference in probabilities of getting y such that $T(y) = 1$ from the two distributions is at most δ.

Let T be any distinguisher, which we view as a Boolean function $T : \{0,1\}^m \to \{1, -1\}$. Notice

that, in this interpretation, $\Pr_{y\to D}[T(y)=1]$ is same as $\mathbb{E}_{y\to D}[T(y)]$. Hence we want to prove that:

$$\left|\mathbb{E}_{y\to D}[T(y)] - \mathbb{E}_{y\to U}[T(y)]\right| \leq \epsilon 2^{m/2}$$

Our starting information is that D is ϵ-biased. That is, by definition, $\left|\mathbb{E}_{y\to D}[\chi_S(y)]\right| \leq \epsilon$. To proceed with the proof, we apply use the Fourier representation of $T = \sum_{S\subseteq[m]} \widehat{T}(S)\chi_S$.

$$\begin{aligned}
\left|\mathbb{E}_{y\to D}[T(y)] - \mathbb{E}_{y\to U}[T(y)]\right| &= \left|\mathbb{E}_{y\to D}\left[\sum_{S\subseteq[m]} \widehat{T}(S)\chi_S(y)\right] - \mathbb{E}_{y\to U}\left[\sum_{S\subseteq[m]} \widehat{T}(S)\chi_S(y)\right]\right| \\
&= \left|\sum_{S\subseteq[m]} \mathbb{E}_{y\to D}\left[\widehat{T}(S)\chi_S(y)\right] - \sum_{S\subseteq[m]} \mathbb{E}_{y\to U}\left[\widehat{T}(S)\chi_S(y)\right]\right| \\
&= \left|\sum_{S\subseteq[m]} \widehat{T}(S)\left[\mathbb{E}_{y\to D}[\chi_S(y)] - \mathbb{E}_{y\to U}[\chi_S(y)]\right]\right| \\
&= \left|\sum_{S\subseteq[m]} \widehat{T}(S)\left[\mathbb{E}_{y\to D}[\chi_S(y)]\right]\right| \qquad \because \mathbb{E}_{y\to U}[\chi_S(y)] = 0 \\
&\leq \epsilon\left|\sum_{S\subseteq[m]} \widehat{T}(S)\right| \leq \epsilon \sum_{S\subseteq[m]} \left|\widehat{T}(S)\right|
\end{aligned}$$

Now we just need to bound $\sum_{S\subseteq[m]} \left|\widehat{T}(S)\right|$. This is just the consequence of the relation between ℓ_2 norm and ℓ_1 norm that we have seen earlier. Consider the 2^m-tuple $\left[\widehat{T}(S)\right]_{S\subseteq[m]}$ as a vector in $\mathbb{R}^{2^m}$. The above expression that we want to bound is the ℓ_1 norm of this vector (sum of absolute values of individual entries). We use the relation between the ℓ_1 norm and ℓ_2 norm.

$$\sum_{S\subseteq[m]} \left|\widehat{T}(S)\right| \leq 2^{m/2} \sum_{S\subseteq[m]} \left[\widehat{T}(S)\right]^2 = 2^{m/2}$$

where the last step is from the Pareseval's identity. Hence the proof. We conclude that:

$$\left|\mathbb{E}_{y\to D}[T(y)] - \mathbb{E}_{y\to U}[T(y)]\right| \leq \epsilon 2^{m/2}$$

□

Now we derive the corollary for almost k-wise independence if ϵ-biased distributions. It is just to observe that the above proof works even for subset of bits of the string y.

COROLLARY 11.6.5. *Any k bits of an ϵ-biased distribution are $\epsilon 2^{k/2}$-close to a uniform distribution on those k bits.*

Let us also get a sense of the parameters. Suppose we want δ-close to a k-wise independent distributions (on every subset of k bits), then we should choose an ϵ-biased distribution such that $\epsilon = \frac{\delta}{2^{k/2}}$. The number of random bits required for this, using our earlier construction is, $2\log m + 2\log\left(\frac{1}{\epsilon}\right)$ (since the size of the set that we constructed earlier is $O\left(\frac{m^2}{\epsilon^2}\right)$. In terms of δ, this is - $2\log m + k + 2\log\left(\frac{1}{\delta}\right)$. Viewing it as a generator, $G : \{0,1\}^\ell \to \{0,1\}^m$ - if we need $G(U_\ell)$ to be k-wise independent, then the best construction we know (and we need) to use $\ell = k\log m$ bit long seed, where as if we need $G(U_\ell)$ to be only "almost" k-wise independent (δ-close on each k bits to uniform), then we can do that with a smaller $\ell = O\left(k + \log m + \log\left(\frac{1}{\delta}\right)\right)$ number of bits as the pure random bit seed for the generator.

Exercise 11.6.6 (See Problem Set 1 (Problem 4))**.**] A k universal set $S \subseteq \{0,1\}^n$ has the property that the projection of S onto any k indexes contains all 2^k possible patterns. Use the previous question to construct k-universal sets of size $\left(2^k \log n\right)^{O(1)}$.

Fooling Space Bounded Computation : Nisan's Generator

Lecturer : Jayalal Sarma

WEEK
12

In this week, we study how to use the objects that we have seen so far in order to design pseudorandom generators for space bounded computation with surprisingly small seed length. We will do a quick review first.

Review of the Model for Space Bounded Algorithms: Space bounded algorithms are formally modelled as follows. The Turing machine model - has a finite control (with a finite number of states), a read-only tape in which input is written, a read-write work-tape in which symbols can be read or written. The space used by the Turing machine is the maximum value of the head position in the worktape at any point during the computation [28].

More formally, a Turing machine is said to be using at most $s(n)$ space if it uses

[28] even if work tape head moves without writing, that is still considered to be space used, since otherwise the head position can be used as additional unapproved storage

Introduction to Error Correcting Codes

Lecturer : Jayalal Sarma

WEEK
13

Recall the discussion about error correcting codes that we did in the beginning of the course. This based on modelling the channel as a probability of flipping every bit.

In this lecture, we will shift our focus from Shanon's viewpoint of codes to Hamming's perspective. The main change is that the channel is modelled using its worst case behavious as opposed to probablistic model that we did earlier.

13.1 Combinatorial View of Codes & Parameters

We clarify the model for the channel now. The channel sends bits in blocks of length n each, and we are given the guarantee that the channel will flip at most t of them in each block, but of course, we do not know which t of the n bits. We will assume that each block is marked out and the guarantee is only within a block. Hence n will be called the *block length* from now on. For natural reasons, we will call k to be the *message length*[29].

Thus, we would like to design codes in such a way that under the guarantee that the channel makes at most t errors, the encoding and decoding functions work correctly. To study the combinatorial properties of error-correcting codes which also leads to constructions, we will begin by introducing the basic framework and crucial parameters of a code.

DEFINITION 13.1.1 (**Error Correcting Codes**). *An error correcting code C of length n over a finite alphabet Σ is a subset of Σ^n. The elements of C are referred to as codewords in C. Recall from the previous lecture that the encoding function of the code is $E : \{0,1\}^k \to \{0,1\}^n$ and the deconding function $D : \{0,1\}^n \to \{0,1\}^k$.*

As done in the previous lecture, we will model the error made by the channel by a bit vector $\eta \in \{0,1\}^n$ which represents the characterestic vector. This gives the following natural definition.

DEFINITION 13.1.2 (**Error Correction**). *For an integer $t \geq 0$, the code $C \subseteq \Sigma^n$ is said to be t-bit error-correcting if there exists a decoding function D such that $\forall m \in \{0,1\}^k, D(E(m) + \eta) = m$ where $\eta \in \{0,1\}^n$ is of weight at most t.*

A related notion that we will define is that of *error detection*. For an integer $t \geq 0$, the code $C \subseteq \Sigma^n$ is said to be t-bit error-detecting code if there exists an error detection algorithm D which for every $m \in \{0,1\}^k$ and error vector $\eta \in \{0,1\}^n$ of weight at most t, outputs "YES" if $E(m) + \eta \in C$ and "NO" otherwise. This notion is useful in situations where the receiver can detect that error

[29] We will also call it the dimension of the code, but for reasons that comes later in the lectures.

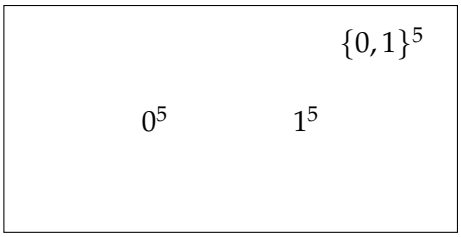
has occured and ask the sender to send the message (equivalently, the codeword) again.

Example 1 : Repeatition Code: Suppose we want to send one bit ($k = 1$) across a channel and we have block length of 5 and $t = 2$. A very easy code to design is the repetition code - which is described by the following encoding function. $E(1) = 11111$ and $E(0) = 00000$.

How many errors can this code correct? Indeed, if the number of bit flips is at most 2, we can still decode. The decoding algorithm is as follows : given a $y \in \{0,1\}^5$ as the received word, output the majority of the 5 bits in the block. It is easy to see that if the original bit was 0 and if $\eta \in \{0,1\}^5$ represents the error bitstring, then $E(0) + \eta$ will have at most two 1s, and hence the decoder will output a 0. The argument is similar for 1 as well.

How many errors can this code detect? If the number bit flips is at most 4, we will be able to detect an error. The detection algorithm simply outputs YES if all the bits are different and NO otherwise. The correctness of the algorithm is immeidate.

Notice that the hamming distance between these two words is 5. As discussed above, the code is 2-bit error correcting and 4-bit error detecting. This was possible because we placed the two codewords very "far away", as a part of the code design.

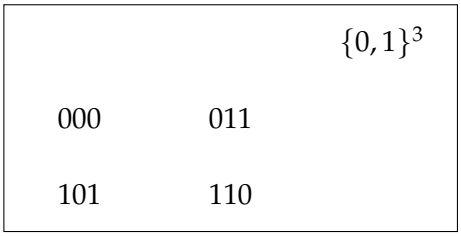


Example 2 : Parity Check Bit Code: Here we will go to the other extreme and add very few redundant bits. Let $n = k + 1$ be the block length. For $m \in \{0,1\}^k$, where $m = (m_0, m_1, \ldots, m_{k-1})$, define $E(m) = \left(m_0, m_1, \ldots, m_{k-1}, \left(\sum_{i=0}^{k-1} m_i\right)\right)$ where addition is modulo 2. We have seen this code in various engineering contexts in the form of checksum bit. This describes the code.

How many errors can this code correct? Unfortunately none. How do we prove this? We can show this by showing a counter example where the received word makes one bit error from the code that was sent and it still has two codewords at equal (hamming) distance from it. Let $k = 2$ and consider message 00 which gets encoded to 000 and consider the received word $y = 001$ where the third bit gets corrupted.. But then the same y could have been obtained from the message 01 which gets encoded to 011, and the second bit corrupted to get the same y !. Hence no decoding algorithm can exist which disambiguates between the messages $m = 00$ and $m' = 01$ when the received word is $y = 001$.

How many errors can this code detect? Yes, it can detect 1 bit errors. The error detection algorithm is as follows. Given a received word $y = (y_0, y_1, \ldots y_k) \in \{0,1\}^n$, output YES if $\sum_i y_i = 1$. Correctness follows from the observation that for all valid codewords in this code, the sum of the bits must be 0 modulo 2.

In the picture in the right side, we take $k = 2$ and $n = 3$. As discussed above, the code is 1-bit error correcting code and cannot correct any error at all. We placed the 4 codewords not so "far away", as a part of the code design. Hence we could pack in a lot of code words for the same n. In other words, it is a very "redundancy efficient" code if we compare the value of n for a given k.



The above two examples demonstrate what we need to worry about while designing the code. We need to keep the codewords far-enough so that we can correct many errors but still close enough to pack in many codewords (2^k) of them within $\{0,1\}^n$ for a given n. These are two contradictory requirements and ideal for both should not be achievable (we will discuss later about the bounds for these). To capture this discussion formally, we define two important parameters associated with a code : the *distance* and *rate*.

DEFINITION 13.1.3 (**Rate of the Code**). *For a code C, the rate $R(C)$ is defined by $R(C) = \frac{k}{n}$. The notion of rate measures the amount of non-redundant information per bit in the codewords of C.*

DEFINITION 13.1.4 (**Minimum Distance of a code**). *The distance of a code C, denoted as $d(C)$, is the minimum Hamming distance between two distinct codewords of C. $\Delta(C) = \min_{x,y \in C; x \neq y} \Delta(x,y)$ The relative distance of C, denoted as $\delta(C)$, is the normalized value $\frac{d(C)}{n}$. Thus, any two distinct codewords of C differ in at least a fraction of δ positions.*

Ideally, we would aim at placing code words as far as possible without blowing up n and achieve good rate while fighting against the channel to correct errors. We denote the specification of a code C defined over alphabet Σ with $|\Sigma| = q$ as $(n,k,d)_q$ to mean that C is a code where codewords are n-tuples of symbols over the alphabet Σ and messages are k-tuples of symbols over the alphabet Σ, such that the distance is at least d for any two codewords in C. Now that the parameters are formally defined, we will capture what we learned from the above two example codes.

LEMMA 13.1.5. *For a code, the following are equivalent:*

1. *Distance is $2t+1$*
2. *It is possible to detect upto $2t$ errors.*
3. *It is possible to correct upto t errors.*

Proof.

5: Jayalal says: Todo - We had done this argument in class, yet to fill in here. This is where the maximum likelyhood decoding algorithm will need to come in

□

Exercise 13.1.6 (**Hash Functions: An Application of Codes** - See Problem Set 3 (Problem 1))**.** A classic data structure problem is to map a large universe U to a (possibly) smaller domain Σ. We

call a hash function $h : U \to \Sigma$ as *good* if $h(x) \neq h(y)$ for all $x \neq y \in U$ (if not, we call it a collision). Indeed, this requires that $|\Sigma| \geq |U|$ and hence is impossible. An interesting variant is to relax on the collision requirement - define a family of hash functions and then argue that for any pair of elements from the universe U, the probability of collision is small for a hash function chosen randomly from the family. More formally, a family of functions from U to M, denoted by $H = \{h_1, h_2, \ldots h_n\}$ is a ϵ-good hash family if for every $x \neq y \in U$, $\Pr_{h \in H}[h(x) = h(y)] \leq \epsilon$.

The task is to construct explicitly, ϵ-good families of hash functions of small size. In this problem, we will use codes with large distance to construct ϵ-good hash families. In fact, we show that the two tasks are equivalent.

If $|\Sigma| = q$, given an $(n, k, d)_q$ code, we define a family of hash functions as follows:

$$H_C = \left\{ h_i : \Sigma^k \to \Sigma \mid 1 \leq i \leq n, \forall x \in \Sigma^k, h_i(x) \stackrel{\text{def}}{=} C(x)_i \right\}$$

(a) Prove that if C is an $(n, k, \delta n)_q$ code, then H_C is $(1 - \delta)$-good. Comment on the size of the family.

(b) Defining the code from the hash family appropriately, show that if we have a hash family which is ϵ-good, then we also get a code from it which is $(n, k, (1 - \epsilon)n)_q$.

13.2 Singleton Bound

We had observed intuitively that it should not be possible to achieve high rate and high relative distance for a code since they seem to be contradictory requirements. Now we will prove a fundamental combinatorial connection between the two.

THEOREM 13.2.1 (**Singleton Bound**). *Any code $(n, k, d)_q$ must satisfy:*

$$d \leq n - k + 1$$

Proof. Let E be the encoding function of the code. $E : \{0,1\}^k \to \{0,1\}^n$. Consider the first $k - 1$ bits of the codewords. Since the original message is k bits in length (for which there are 2^k possible messages, but the first $k - 1$ bits can only be $2^{k-1} < 2^k$ in number, by pigeon hole principle, there exists atleast messages $m, m' \in \{0,1\}^k$ such that the first $k - 1$ bits agree for $E(m)$ and $E(m')$. But then, these two code words can differ in maximum of $n - (k - 1)$ bits and hence the minimum distance d is at most $n - k + 1$. □

Dividing by n on both sides, $\frac{d}{n} \leq 1 - \frac{k}{n} + \frac{1}{n}$. Hence, $\delta \leq 1 - R + \frac{1}{n}$. Thus, if the rate of the code is high then the relative minimum distance must be small. Or the two should add up to almost 1. This gives us an intuitive way of the connection between the rate and the distance. Notice that the above statement is independent of the size of the alphabet. A natural question is to ask whether there are codes which achieves the singleton bound? It turns out that there are, but they have non-constant alphabet size. Even more interestingly one can argue that singleton bound cannot be achieved with constant alphabet size (not just two !). We will take this up for discussion in the next lecture.

In conclusion, one can see two combinatorial objectives while constructing codes. Ideally, we would like to come up with a $(n, k, d)_q$-code that

(a) maximizes d for a given n, k and q to achieve best possible error correcting capability (and with efficient encoding and deconding algorithms, whereever possible).

(b) minimizes n for a given k, d and q to achieve maximum rate for a fixed d (and with efficient encoding and deconding algorithms, whereever possible).

13.2.1 The Hamming Code

Now that we have more motivation to construct linear codes with good parameter (as clarified by the above exercise question). Do we have distance 1 codes., yes the trivial identity function works for rate 1 and distance 1. The case for $d = 2$ is our parity checkbit code.

We now attempt on $d = 3$. We study a family of codes constructed by Hamming, where the rate of the code is very close to optimal but indeed, as implied by singleton bound suffers from very low distance (in fact, distance is 3 and hence can correct only 1 bit errors). The idea is to encode every 4-bit information to a 7-bit codeword. Consider $m = x_1, x_2, x_3, x_4$. Now, $E : \{0,1\}^4 \to \{0,1\}^7$ described as follows :

$$(x_1, x_2, x_3, x_4) \longmapsto (x_1, x_2, x_3, x_4, x_2 \oplus x_3 \oplus x_4, x_1 \oplus x_3 \oplus x_4, x_1 \oplus x_2 \oplus x_4)$$

Notice that the arithemetic is modulo 2 and hence we should think of $E : \mathbb{F}_2^4 \to \mathbb{F}_2^7$. The parameters of the code are not clear directly from the construction and hence we prove the following lemma.

PROPOSITION 13.2.2. *Hamming Code for $k = 4$ has rate $R = 4/7$ and distance 1.*

Proof. The rate expression is immediate. For arguing the distance, we observe the following. Consider $c, c' \in \mathbb{F}_q^n$ are codewords, then their bitwise sum $c + c'$ is also a codeword. Indeed, if m and m' are messages such that $E(m) = c$ and $E(m') = c'$. Let Δ denote the hamming distance between two words. Notice that the distance d is :

$$d = \min_{\substack{m,m' \in \mathbb{F}_q^n \\ m \neq m'}} \Delta(E(m), E(m')) = \min_{\substack{m,m' \in \mathbb{F}_q^n \\ m \neq m'}} |E(m) + E(m')| = \min_{\substack{m,m' \in \mathbb{F}_q^n \\ m \neq m'}} |E(m + m')| = \min_{\substack{m \in \mathbb{F}_q^n \\ m \neq 0}} |E(m)|$$

Thus the minimum distance of the code is the minimum weight of any non-zero codeword in the code. Now we show that this is exactly 3 for Hamming code.

CLAIM 13.2.3. *Weight of any non-zero codeword in Hamming code is at least 3, and there is a codeword of weight 3.*

Proof. Indeed there is a codeword of weight 3. Consider the message 1000, this gives rise to the codeword 1000011. To show the lower bound, consider any message $m = (x_1, x_2, x_3, x_4) \neq 0000$. We have the following cases:

$wt(m) = 1$: We observe the peculiarity that for any bit x_i, the number of redundant bits that is

contributes to is at least 2 for any i (in fact, it is exactly 2 for all except x_4 and is three for x_4. Hence the weight of the codeword when the message is of weight 1 is at least three.

$wt(m) = 2$: For this, we have to observe that for x_i, x_j, there is exactly at least one redundant bit to which x_i contributes and x_j does not contribute or vice versa. Hence when both x_i and x_j are 1s and the rest of the x_i's is zero, this redundant bit also be 1 and will contribute to the weight of the word. Hence the weight will be at least 3.

$wt(m) \geq 3$: This is trivial since the message itself is a part of the codeword and hence the codeword will have weight at least 3.

While the above argument completes the proof, you might find it being far from being elegant since it might look very adhoc. Fortunately, there is a more linear algebraic argument for the above theorem which we will present in the next section. □

□

Now that the distance is $d = 3$, we will ask the usual questions about how many errors can it correct and how many errors can it detect and that too how efficiently. Indeed, the distance of the code is 3, hence the maximum likelyhood decoding will be able to correct up to 1 bit error. However, this decoding is far from efficient since it will have run through different codewords to figure out which one the received word is closest to. It is unclear how to do error detection for up to 2 errors efficiently (the trivial algorithm again is to run over the possible codewords).

13.3 Linear Codes

One of the main steps in the above distance arugment was the following observation. For Hamming code, if m and m' are two messages over $\mathbb{F}_q^k$, then for any $\alpha, \beta \in \mathbb{F}_q$, $E(\alpha m + \beta m') = \alpha E(m) + \beta E(m')$ which is a linearity condition. We call such codes to be linear codes. The repetition code, the parity check bit code, the Hamming code are all linear by design. Recall that a subset C of $\mathbb{F}_2^n$ is a linear subspace if (1) $0^n \in C$ (2) $c, c' \in C \Rightarrow c + c' \in C$ (3) $c \in C \Rightarrow \alpha c \in C$ for each scalar $\alpha \in \mathbb{F}_2$. Thus, if $\Sigma = \mathbb{F}_q$ is a field and $C \subset \Sigma^n$ is a subspace of Σ^n then C is said to be a linear code.

A more important information is that since the code C is linear subspace, the encoding process can be thought of as a linear transformation $E : \mathbb{F}_2^k \mapsto \mathbb{F}_2^n$ whose range is exactly C. Let G be the $k \times n$ matrix representing this linear transformation such that for $x \in$
F_q^4 viewed as a row vector, $E(x) = xG$. Here, $x = [x_1 x_2 x_3 x_4]$ and G is the matrix

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

This matrix is called the generator matrix of the code. We define this parameter for linear codes in general.

DEFINITION 13.3.1 (**Generator Matrix of a Code**). *For a linear code $C \subseteq \mathbb{F}_q^n$, with k as the message length, the generator matrix of the code is the $k \times n$ matrix, $G \in \mathbb{F}_q^{k \times n}$ such that C is the row space of G.*

$$C = \{xG \mid x \in \mathbb{F}_q^k\}$$

Hence, the k rows of the matrix G has to be linearly independent vectors in $\mathbb{F}_q^n$ and forms a basis of the subspace C. Due to this, k is also sometimes called the dimension of the code C.

As for notation, we will represent linear codes with the parameter n, k, d, q as $[n, k, d]_q$ code.

Exercise 13.3.2 (**Combining Codes** - See Problem Set 3 (Problem 2))**.** Let C_1 be an $[n_1, k_1, d_1]_2$ binary linear code, and C_2 be an $[n_2, k_2, d_2]$ binary linear code. Let $C \subseteq \mathbb{F}_2^{n_1 \times n_2}$ be the subset of $n_1 \times n_2$ matrices whose columns belong to C_1 and whose rows belong to C_2. Show that C is an $[n_1 n_2, k_1 k_2, d_1 d_2]_2$ binary linear code.

For $w \in \mathbb{F}_q^n$, $|w| = |\{\ i \mid w_i \neq 0\ \}|$. From the above discussion, we have the following proposition:

PROPOSITION 13.3.3. *If $C \subseteq \mathbb{F}_q^n$ is an $[n, k, d]_q$ code, then: $d = \min_{\substack{m \in \mathbb{F}_q^n \\ m \neq 0}} |E(m)|$*

Proof. The argument just uses linearity and is same as what we used earlier.

$$d = \min_{\substack{m, m' \in \mathbb{F}_q^n \\ m \neq m'}} \Delta(E(m), E(m')) = \min_{\substack{m, m' \in \mathbb{F}_q^n \\ m \neq m'}} |E(m) + E(m')| = \min_{\substack{m, m' \in \mathbb{F}_q^n \\ m \neq m'}} |E(m + m')| = \min_{\substack{m \in \mathbb{F}_q^n \\ m \neq 0}} |E(m)|$$

□

This gives us a way to compute distance of linear codes and makes it easy to prove the distance for the parity check code and hamming code easily. Notice that the encoding algorithm is very efficient since it can now be thought of as computing a linear transformation, which can be done in $O(kn)$ time. The interesting question to ask is decoding. While the maximum likelyhood decoding works as always (although inefficient), one might dream of designing efficient decoding algorithms for all linear codes ideally (exploting the linearity structure). However, this far from what we have now.

We start with a simpler task. Can we detect errors efficiently? Notice that this reduces to an efficient recognition of codewords. We now develop the machinery for that. Recall that every linear subspace $C \subseteq \mathbb{F}_q^n$ of dimension k, has an orthogonal space defined as follows:

$$C^\perp = \{w \in \mathbb{F}_q^n \mid \forall c \in C, \langle c, w \rangle = 0\}$$

We can verify that $C^\perp$ is a subspace and has dimension of $C^\perp$ is $n - k$. The set $C^\perp$ can be viewed as a code by itself (and we call it the *dual* code of C). Let H be the generator matrix of the code $C^\perp$. The rows of H will form a basis for $C^\perp$ and hence if $c \in C$ where $r_1, r_2, \ldots r_k \in \mathbb{F}_q^n$ forms the basis for C (rows of G) and hence $c = \sum_i \alpha_i r_i$. Since rows of H, $w_1, w_2, \ldots w_{n-k}$ forms the basis for $C^\perp$ (and hence in $C^\perp$), we have that $Hc = H\left(\sum_i \alpha_i r_i\right) = \sum_i \alpha_i H r_i = 0$. This gives the following definition:

DEFINITION 13.3.4 (**Parity Check Matrix of a Code**). *Parity check matrix of a linear code $C \subseteq \mathbb{F}_2^n$ is a $(n-k) \times n$ matrix H such that*

$$c \in C \iff Hc^T = 0.$$

LEMMA 13.3.5. *For any code C : H is a parity check matrix and G as the generator matrix if and only if $GH^T = 0$.*

Proof. The proof almost follows from the above discussion.

$$\begin{aligned} c \in C &\iff c \in \mathsf{Row\text{-}Span}(G) \iff Hx^T = 0 \\ &\iff H \cdot G^{(i)} = 0, \forall\, i \in \{1, \ldots, k\},\ G^{(i)} \text{ is the ith row of } G \\ &\iff HG^T = 0 \iff GH^T = 0. \end{aligned}$$

□

We will write down the parity check matrix of Hamming code explicitly. Consider $m = x_1, x_2, x_3, x_4$. By the Hamming code encoding, $E(m) = y_1, y_2, y_3, y_4, y_5, y_6, y_7$ where $y_i = x_i$ for $1 \leq i \leq 4$ and $y_5 = x_2 \oplus x_3 \oplus x_4$, $y_6 = x_1 \oplus x_3 \oplus x_4$ and $y_7 = x_1 \oplus x_2 \oplus x_4$. A word $c \in \{0,1\}^7$ is a valid codeword if it satisfies the following equalities:

$$\begin{gathered} c_5 \oplus x_2 \oplus x_3 \oplus x_4 = 0 \\ c_6 \oplus x_1 \oplus x_3 \oplus x_4 = 0 \\ c_7 \oplus x_1 \oplus x_2 \oplus x_4 = 0 \\ c_i = x_i \text{ for } 1 \leq i \leq 4 \end{gathered}$$

In matrix terms, this is equivalent to saying that $y' \in \{0,1\}^7$ is a valid codeword if

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} c_7 \\ c_6 \\ c_1 \\ c_5 \\ c_2 \\ c_3 \\ c_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}^T$$

Thus, we have an immeidate 2 bit error detection and 1-bit error correction for Hamming codes. For detecting whether there was error (if at most 2 bits of error happened) for a received word, check if $Hy^T = 0$. To correct 1-bit error, given transmitted word y, consider the n ways of at most 1-bit flip and check if any of them is in the nullspace of H or not. Thus, by at most n matrix multiplications, the single bit error can be corrected.

All this is good for Hamming code. For a general linear code, how do we construct parity check matrix of a code from the generator matrix?

LEMMA 13.3.6. *Let $G \subseteq \mathbb{F}_2^{k \times n}$ be a generator matrix, where*

$$G = [I_k \mid P]$$

then

$$H = \left[-P^T \mid I_{n-k}\right]$$

is a parity check matrix for C.

Proof. It is easy to verify that $GH^T = 0$, then by Lemma 13.3.5 the result follows. □

Going back to the case of Hamming codes, recall the above error correction algorithm, which took n linear tranformation computations in the worst case. Because of the special design of Hamming codes, this can be done directly by one linear transformation. Suppose that we have a received word y with the guarantee that at most one bit (say $i \in [n]$) was corrupted. Let e_i be the column vector of zeros in all positions but i. Now, if we represent y as $c + e_i$, where c is the codeword sent and y is the received word. Note that $Hy = H(c + e_i) = Hc + He_i = He_i$. He_i is the i^{th} column of H denoting the binary representation of i. Thus, the erroneous bit i is corrected by a single multiplication.

Hamming Code packs in maximum number of codewords: Notice that one of the aims of the code design is to maximise $|C|$ given d and n. How may words can be packed in if the minimum distance is restricted to 3. We answer this first:

LEMMA 13.3.7. *If C is a $(n, k, 3)_2$-code, then $|C| \leq \frac{2^n}{n+1}$.*

Proof. For each $x \in C$, define $N(C) = \{y \in \{0,1\}^n \mid \Delta(x, y) \leq 1\}$. Since $d = 3$, $N(x) \cap N(y) = \emptyset$ for $x \neq y \in C$. Therefore, $2^n \geq |\bigcup_{x \in C} N(x)| = \sum_{x \in C} |N(x)| = |C| \cdot (n + 1)$. □

COROLLARY 13.3.8. *For any word $w \in \{0,1\}^7$, there is a codeword at a distance of at most 1 in the Hamming code.*

Proof. In the Hamming code C, if $n = 4$ then $k = 3$. Further, there are 16 distinct codewords in C. Each codeword has 7 possibilities of 1-bit flip. Thus, by Lemma 13.3.7, the upper bound on $|C|$ is achieved. Hence, the Hamming balls of radius 1 around the codewords have empty pair-wise intersection. Also, the union of the set of words in the balls is precisely $\{0,1\}^7$. Therefore, every $w \in \{0,1\}^7$ has a codeword at a distance of at most 1. □

Exercise 13.3.9 (From Linear Codes to k-wise Independence - See Problem Set 3 (Problem 3)**).** For integers $1 \leq k \leq n$, call a subset $S \subseteq 0,1^n$ to be k-wise independent if for every $1 \leq i_1 < i_2 < \cdots < i_k \leq n$ and $a \in 0,1^k$

$$\Pr_{x \in S}[x_{i_1} = a_1 \wedge x_{i_2} = a_2 \wedge \cdots \wedge x_{i_k} = a_k] = \frac{1}{2^k}$$

where the probability is over an element x chosen uniformly at random from S. In this problem, you will see how codes can be used to construct k-wise independent sets of small size. Recall the constructions that we have seen earlier.

(a) Let $H \in \mathbb{F}_2^{t \times n}$ be the parity check matrix of an $[n, n - t, d]_2$ binary linear code of distance

$d > k+1$. Define:

$$S = \left\{ x^T H \mid x \in \mathbb{F}_2^t \right\}$$

Prove that S is a k-wise independent set of size 2^t.

(b) Using the above, show how one can construct a k-wise independent subset of $\{0,1\}^n$ of size at most $(2n)^{(k/2)}$.

We start by proving the generalization of the sphere packing argument that we presented in the last lecture.

13.4 Hamming Bound

LEMMA 13.4.1 (**Hamming Bound**). *For an (n,k,d) code C,*

$$|C| \leq \frac{2^n}{\sum_{i=1}^{\lfloor \frac{d-1}{2} \rfloor} \binom{n}{i}}$$

Proof. Let d be the minimum distance of the code C. Consider the Hamming balls of radius $\lfloor \frac{d-1}{2} \rfloor$ centred at the codewords. The number of strings in $\{0,1\}^n$ within this distance is at most $\sum_{i=1}^{\lfloor \frac{d-1}{2} \rfloor} \binom{n}{i}$. None of these balls intersect with each other, since otherwise the minimum distance will be less than d. Thus each word is covered at most once (we may skip some). Thus if we count the "volume" of each of the Hamming balls:

$$|C|. \sum_{i=1}^{\lfloor \frac{d-1}{2} \rfloor} \binom{n}{i} \leq 2^n$$

The Hamming bound follows. □

An intuitive interpretation of Hamming Bound is as a tradeoff result between n and k. It shows the limits on on packing too many codewords in the space $[0,1\}^n$ while keeping the minimum distance to be large. Thus, for a given minimum distance if we want to pack in too many codewords, we have no option but to increase the value of n. Codes which achieve the Hamming bound are called *Perfect Codes*.

13.5 Generalized Hamming Codes

Now we generalize the construction. Let n be of the form $2^r - 1$. We specify the code by describing the parity check matrix. The parity check matrix of the Hamming code is the matrix formed by all r bit representations of the numbers from $1, 2, \ldots 2^r - 1$. Note that this excludes all 0s. Hence the parity check matrix is of dimensions $(2^r - 1) \times r$ which is $(n-k) \times n$. To calculate the distance, we use the fact that the minimum distance of a linear code is the size of the smallest subset of columns of H which are dependent (Prove this in exercise). The first three columns of the matrix are linearly dependent, and no two columns are linearly dependent. Hence the Hammind Code

is an $[2^r - 1, 2^r - r - 1, 3]_2$ code. Notice that the decoding procedure works exactly as the case for $k = 3$ case.

We check the Hamming bound. We know that $|C| = 2^{2^r - 1 - r}$. Thus, $|C| = \frac{2^{2^r-1}}{2^r} = \frac{2^n}{n+1}$. We observe that any general hamming code of length $2^r - 1$ is also a perfect code.

Exercise 13.5.1 (**The Hat Game** - See Problem Set 5 (Problem 4))**.** Consider a game involving n people in a room, each of whom is given a red/white hat chosen uniformly and independently at random. Each person can see the hat color of all other people, but not their own. They are asked to guess their own hat color which they can decline or make a guess (simultaneously). They either win collectively, or lose collectively. Once the guessing is done, they win if no body makes a wrong guess and at least one person makes the correct guess. They cannot interact during the game, but can have some predetermined strategy among themselves.

A trivial strategy is to make only one person guess and everyone else decline. This gives a probability of $\frac{1}{2}$. In this problem, our goal is to come up with a much better strategy. The problem presents an interesting application of Hamming Codes.

A directed graph G is a subgraph of the n-dimensional hypercube if its vertex set is $\{0,1\}^n$ and if (u, v) is an edge in G, then $\Delta(u, v) = 1$ (hamming distance).

For a graph G, let $K(G)$ be the number of vertices of G with in-degree at least one, and out-degree zero.

(a) Show that: $\Pr\{\text{Winning Hat Game}\} = \max_G \frac{K(G)}{2^n}$ (Hint : Establish a bijection between strategies and subgraphs. Think of the endpoints of an edge as two possible views for a player.)

(b) Using the fact that the out-degree of any vertex is at most n, show that, for any directed subgraph G of the n-dimensional hypercube :

$$\frac{K(G)}{2^n} \leq \frac{n}{n+1}$$

(c) Use hamming codes to show that if $n = 2^\ell - 1$ then there exists a directed subgraph of G of the n-dimensional hypercube with

$$\frac{K(G)}{2^n} = \frac{n}{n+1}$$

13.6 Dual Codes and Hadamard Code

Recall the definition of the dual of a code defined by the orthogonal space to the code subspace. That is, if $C \subseteq \mathbb{F}_q^n$,

$$C^\perp = \{w \in \mathbb{F}_q^n \mid \forall c \in C, \langle c, w\rangle = 0\}$$

If we view $C^\perp$ as a code itself, we get the dual code of C. The generator matrix of C is the parity check matrix of $C^\perp$ and vice versa. Indeed, if C is an $[n, k]_q$ code, then $C^\perp$ is an $[n, n-k]_q$ code.

As our first example, we will think about the dual code of the Hamming code itself. This is easy since we have described the Hamming code directly by its parity check matrix itself (whose columns are the set of all r-bit representations of he numbers $1, 2 \ldots 2^r - 1$. Viewing this as a generator matrix will give us the *simplex code* which is $[2^r - 1, r, 2^{r-1}]_2$ code. If we add the all 0s

column to the generator matrix, then we get a special code called the *Hadamard code* which is a $[2^r, r, 2^{r-1}]_2$ code. We quickly prove the claim about the distance.

Exercise 13.6.1. For the paramter r used above, Hadamard code and Simplex code both have distance 2^{r-1}. In fact something stronger is true, every non-zero codeword has weight exactly 2^{r-1}.

The Hamming and Hadamard codes show the two ends of a spectrum. Hamming code has rate $\frac{2^r-r-1}{2^r-1}$ (which is close to 1) while Hadamard code has rate $\frac{r}{2^r}$ (which is close to 0). On the distance front, the Hamming code has very low distance 3 (independent of r) and Hadamard code has distance 2^{r-1}.

Exercise 13.6.2. Let $G = (V, E)$ be any undirected graph (assume no loops or multiple edges). A cut in the graph is the subset of all edges that connect a vertex in S to vertex in $V \setminus S$, for some subset $S \subseteq V$. Let $\mathsf{Cuts}(G) \subseteq \{0,1\}^E$ consist of the characteristic vectors of all cuts of G.

(a) Prove that $\mathsf{Cuts}(G)$ is an $[|E|, |V| - 1]_2$ binary linear code. What parameter of G equals the distance of Cuts(G)?

(b) Describe the dual code $\mathsf{Cuts}(G)^\perp$ of $\mathsf{Cuts}(G)$. What is its dimension? What parameter of G equals the distance of $\mathsf{Cuts}(G)^\perp$?

Exercise 13.6.3. Assume that there exists an $[n, k, d]_q$-code C over $\mathbb{F}_q$. Then:

(a) (**Extension**) There exists an $[n + r, k, d]_q$ code for every $r \geq 1$.

(b) (**Puncturing**) There exists an $[n - r, k, d - r]_q$ code for every $r \in [d - 1]$.

(c) There exists an $[n, k, d - r]_q$-code over $\mathbb{F}_q$ for every $1 \leq r \leq d - 1$.

(d) (**Subcode**) There exists an $[n, k - r, d]_q$ code for every $r \in [k - 1]$.

(e) There exists a $[n - r, k - r, d]_q$ code for every $r \in [k - 1]$.

(f) (**Direct Sum**) Let C_1, C_2 be linear codes such that for $i \in [2]$, C_i is an $[n_i, k_i, d_i]_q$. Define the direct sum :

$$C_1 \oplus C_2 = \{(c_1, c_2) \mid c_1 \in C_1, c_2 \in C_2\}$$

is an $[n_1 + n_2, k_1 + k_2, \min\{d_1, d_2\}]_q$-code. Write down the generator matrix for the direct sum code.

(g) Let C_1, C_2 be linear codes such that for $i \in [2]$, C_i is an $[n, k_i, d_i]_q$. Define the direct sum :

$$C = \{(c_1, c_1 + c_2) \mid c_1 \in C_1, c_2 \in C_2\}$$

is an $[2n, k_1 + k_2, \min\{2d_1, d_2\}]_q$-code. (Hint: Analyse in two cases : $c_2 = 0^n$, $c_2 \neq 0^n$.)

(h) Let $\overline{x}$ denote the bitwise complement of a vector x, and let C be a $[n, k, d]_2$-code. Then the code:

$$C' = \{(c, c), (c, \overline{c}) \mid c \in C\}$$

is a $[2n, k + 1, \min\{2d, n\}]_2$-code. (Hint : Repetition code) This provides us a way of rovides us a way to double all parameters (n, k and d).

Reed-Solomon Code, Reed-Muller Codes : Construction & Decoding

Lecturer : Jayalal Sarma

WEEK

14

In this lecture, we will describe Reed-Solomon code, a fundamental code in the algebraic coding theory. It is an example of an MDS code (achieves the singleton bound) but with a large alphabet. We also give the unique decoding algorithm for Reed-Solomon Codes.

14.1 Reed-Solomon codes

For any n and $k \leq n$, and $q \geq n$, there is a Reed-Solomon code $[n, k, d]_q$. We describe the code by describing the encoding procedure. The message $m = (m_0, \ldots, m_{k-1}) \in \mathbb{F}_q^k$ can be viewed as a polynomial $p_m(x) = \sum_{i=0}^{k-1} m_i x^i$ of degree $k-1$ over $\mathbb{F}_q$. Fix a subset $\{\alpha_1, \alpha_2, \ldots \alpha_n\}$ of size n for the field $\mathbb{F}_q$.

$$E_{RS}(m) = (p_m(\alpha_1), p_m(\alpha_2), \ldots, p_m(\alpha_n))$$

Reed-Solomon code is indeed a linear code - the generator matrix can be described as follows:

$$G_{RS} = \begin{pmatrix} 1 & 1 & \ldots & 1 \\ \alpha_1^1 & \alpha_2^1 & \ldots & \alpha_n^1 \\ \vdots & & & \\ \alpha_1^{k-1} & \alpha_2^{k-1} & \ldots & \alpha_n^{k-1} \end{pmatrix}$$

We now compute the distance of the Reed-Solomon code.

LEMMA 14.1.1. *For a $[n, k, d]_q$ Reed Solomon code, $d = n - k + 1$.*

Proof. Because of singleton bound, we just need to show that $d \geq n - k + 1$. Since RS codes are linear, we have

$$d = \min_{0 \neq m \in \mathbb{F}_q^k} \mathsf{weight}(E(m))$$

Thus, $d = \min_{m \in \mathbb{F}^k} |\{i \mid p_m(\alpha_i) \neq 0\}| = n - \max_{m \in \mathbb{F}^k} |\{i \mid p_m(\alpha_i) = 0\}|$

Since $p_m(x)$ is a polynomial of degree at most $k-1$, it can have at most $k-1$ roots. In other words, $\max_{m \in \mathbb{F}^k} |\{i \mid p_m(\alpha_i) = 0\}| \leq k-1$, and hence $d \geq n - (k-1)$ and hence $d \geq n - k + 1$. □

One of the disadvantages of Reed-Solomon codes it that it insists that the field size must be at least n. So the next attempt is to achieve (or attempt to achieve) singleton bound with smaller fields, resulting in Reed-Muller codes.

Curiosity 14.1.2 (Concatenation of Codes). A natural question that may arise is why could we not just encode the q-ary alphabet symbols using binary alphabet "inside" the code and then use

it as a binary code. One immediate difficulty with this is that one bit change in the encoding can change the whole symbol - hence 1 bit error can potentially have the effect of $O(\log q)$ errors. This will make the parameters quite weak. Then, how about using another code inside the binary encoding so that we can correct those errors too. While it may look complicated, it is possible do this systematically and construct a code with a small alphabet from a code with a large alphabet. Let q by a prime power, and let C be an $[N, K, D]_{q^k}$ code and let C' be an $[n, k, d]_q$ code . Then there exists a linear code C'' over $\mathbb{F}_q$ with parameters $[nN, kK, d']_q$ such that $d' \geq dD$. C is called the *external* code, and C' is called the *internal* code.

This is particularly interesting to apply for concatenation two codes with good distances. Consider the Reed-Solomon codes which requires large alphabet size, and the Hadamard code. The external code is the Reed-Solomon code $[n, \frac{n}{2}, \frac{n}{2} + 1]$ over $\mathbb{F}_q$ with $q = n = 2^r + 1$ for some r. We then concatenate this code with the Hadamard code with parameters $[2^r, r + 1, 2^r - 1]$. In terms of n the inner code is $[\frac{n}{2}, \log n, \frac{n}{4}]$. This concatenation is legal because the block length of Reed-Solomn code and the message length of Reed-Muller code exactly matches. By the above theorem (in the previous paragraph), we have that the concatenated $[\frac{n^2}{2}, \frac{n}{2} \log n, \frac{n^2}{8}]$. But unfortunately this codes does not have constant rate.

The problem with concatenating Reed-Solomon codes with Hadamard code is that the internal code does not have constant rate. We would therefore like to concatenate the Reed-Solomon code with a internal binary code with constant rate and distance. One can choose codes which achieves the Gilbert-Varshamov bound. But then, we do not know if such a code that is explicitly constructible. Even more interestingly, **[?]** asked *do we require inner code to be explict and efficiently encodable/decodable?* The answer is no, because the codewords are symbols of the external code and the internal codes message lengths are logarithmic in the blocklength. Thus, if the internal code can be constructed in time that is exponential in the message length of the code, then this will still be polynomial in the length of the concatenated code and still good enough for us. The asymptotically good codes (costant rate and relative distance) constructed out of this idea are called Forney Codes **[?]**.

14.2 Reed Muller Codes

Let us start with some efforts at reducing the field size. Note that for Reed-Solomon codes, we need at least n elements in the field since the code tuples must be of length n. So the question is

" Can n tuples be still obtained with a smaller field ? "

The answer is yes and can be achieved by moving to a multivariate polynomial. We shall see how to reduce the field size by the following example of bivariate polynomials.

Let $p(x, y)$ be a bivariate polynomial of degree $\leq t$. Hence,

$$p(x, y) = \sum_{\substack{0 \leq i,j \leq t \\ i+j \leq t}} m_{ij} x^i y^j$$

We shall interpret the message as the coefficients of this polynomial. Indeed, we need to ensure that the number of coefficients match up.

Exercise 14.2.1. For a bivariate polynomial of degree t, the number of monomials given by

$$|\{(i,j)|0 \leq i,j \leq t, i+j \leq t\}|$$

is $\binom{t+2}{2}$. In general show that for a v variate polynomial of degree t, number of monomials is $\binom{t+v}{v}$.

Hence, the message length $k = \binom{t+2}{2}$ or $k = \frac{(t+2)(t+1)}{2} > \frac{t^2}{2}$. Hence, $t < \sqrt{2k}$.

Now to generate n evaluations of $p(x,y)$, the set S needs to have only $\sqrt{n}$ elements thereby reducing the requirement on the field size to $q \geq \sqrt{n}$. Hence for a message $m = (m_1, m_2, \ldots, m_k)$ where $k = \binom{t+2}{2}$,

$$C(\overline{m}) = (p_m(\alpha,\beta))_{\alpha,\beta \in S^2}$$

where p_m is the bivariate polynomial whose coefficients are the m_is with indices interpreted appropriately.

Exercise 14.2.2. Argue that the Reed-Muller code is linear.

Minimum distance of Reed-Muller Codes: Since the code is linear, it suffices to estimate the weight of minimum weight code word. Hence it is sufficient to find the minimum number of non-zero entries in any n-tuple that is a codeword. To obtain this bound we use the following lemma which we have seen in the first lecture in the course.

LEMMA 14.2.3 (**Schwartz-Zippel Lemma**). *Let $p \not\equiv 0$, be a polynomial in r variables $(x_1, x_2, \ldots, x_n)$ of degree $t > 0$ defined on a field $\mathbb{F}$. Let S be a non-empty subset of $\mathbb{F}$. Then,*

$$\mathsf{Pr}_{\overline{a} \in S^r}[P(\overline{a}) = 0] \leq \frac{t}{|S|}$$

Applying the above lemma, we have

$$\mathsf{Pr}_{a \in S^2}(p(\overline{a}) = 0) = \frac{\text{total degree}}{|S|} < \frac{\sqrt{2k}}{\sqrt{n}}$$

Hence, number of zeros in any code word $< \sqrt{\frac{2k}{n}} \times |S|^2 = \sqrt{\frac{2k}{n}} \times n = \sqrt{2kn}$. Therefore # of non zero entries is $> n - \sqrt{2kn}$. By linearity of the code, minimum distance $d > n - \sqrt{2kn}$. Also $d \leq n - k + 1$. Hence,

$$n - \sqrt{2kn} < d \leq n - k + 1$$

This inequality points to the trade-off between the distance and field size for the Reed-Muller code construction.

Now, let us consider the general case where we have a degree t polynomial in v variables. By Schwatrz-Zippel lemma, number of zero $\leq \frac{t}{|S|}.|S|^v = t|S|^{v-1}$. Hence minimum distance $d \geq n - t|S|^{v-1}$. Since $|S|^v = n$, $|S| = n^{1/v}$. Hence:

$$d \geq n - t.n^{\frac{v-1}{v}}$$

and t and k are related as $k = \binom{t+v}{v}$. Here again, we can see the trade-offs: increasing n (for obtain-

ing a larger code word) or increasing v (to reduce the field size) clearly decreases the minimum distance.

Hadamard Code as a Reed-Muller Code: Let polynomial $p(x_1, x_2, \ldots, x_n) = m_1x_1 + m_2x_2 + \ldots + m_kx_k$ where $\mathbf{m} = (m_1, m_2, \ldots, m_k) \in \mathbb{F}_2^k$ is the message polynomial. Encoding would be,

$$C(\mathbf{m}) = (p(a_1, a_2, \ldots, a_n))_{(a_1,a_2,\ldots,a_n)} \in \mathbb{F}_2^n$$

Thus, we obtain all possible combinations of the message which is nothing but a $[2^k, k, 2^{k-1}]$ Hadamard code. Hence Hadamard codes can be seen as special case of Reed-Muller code.

14.3 Decoding Problem

As we observed earlier, the encoding of linear codes is an easy task, but efficient decoding is far from clear.

We recall the (unique) decoding problem for a code C can be formulated as follows:

PROBLEM 14.3.1 (**Unique Decoding Problem**). *The unique decoding problem for codes is as follows:*
Input: *An $[n, k, d]_q$ code C, $t \in \mathbb{N}$, and a word $y \in \mathbb{F}^n$, with $\Delta(y, C) \leq t \leq \frac{d-1}{2}$.*
Output: *Output a codeword c such that $\Delta(c, y) \leq t$.*

Note that, since the number of erros is limited to $t = \frac{d-1}{2}$, there will be a unique codeword within the Hamming distance of t from the given word $y \in \mathbb{F}^n$, since $t \leq \frac{d-1}{2}$. A brute force approach would be to guess t index locations where the target codeword differs from the input word y, and try all possible modifications to y at the locations guessed. This would require $\binom{n}{t}(q-1)^t$ membership queries to C. The brute-froce algorithm worked well in the case of Hamming codes as their distance $d = 3$. In the following section, we study the Unique decoding problem for Reed-Solomon Codes.

14.4 Decoding Reed-Solomon Codes : Berlekamp-Welch Algorithm

The unique decoding problem for Reed-Solomon codes can be re-formulated as follows:
Input: $[n, k, d]_q$, $d = n - k + 1$, $\alpha_1, \ldots, \alpha_n \in \mathbb{F}$ as parameters for the RS-code, $y = (y_1, y_2, y_3, \ldots, y_n) \in \mathbb{F}^n$ with the guarantee that there is a codeword $c \in RS$, with $\Delta(c, y) = t < \frac{n-k+1}{2} \ (= \frac{d}{2})$
Output: A polynomial P of degree $\leq (k-1)$ such that the $|\{i_{\in[n]} \ (P(\alpha_i) \neq y_i)\} \leq t$

A natural algorithm would be to guess a polynomial p of degree at most $k-1$ and then verify if the codeword corresponding to p is at a distance of t from y. This would put the decoding problem in the complexity class NP. A polynomial time algorithm for the problem was first given by Wesley Peterson in 1960. Later on the running time of the algorithm was improved by many authors. We discuss an algorithm by Berlekamp and Welch in full detail. We begin with the notion of error locator polynomials:

DEFINITION 14.4.1 (**Error Locator Polynomial**). *Let C be an $[n.k,d]_q$ RS-code with the evaluation set $S = \{\alpha_1, \ldots, \alpha_n\}$. A polynomial $E(x)$ is an error locator polynomial for a given word $y \in \mathbb{F}^n$, if $\forall i$ $E(\alpha_i) = 0$ iff $p(\alpha_i) \neq y_i$*

LEMMA 14.4.2. *There exists an error locating polynomial E of degree at most t, where $t = \Delta(y, C)$.*[30]

Proof. Suppose P is a polynomial of degree at most $k-1$ such that $\Delta(y, (p(\alpha_1), \ldots, p(\alpha_n)) = t$. Let $T = \{i \mid P(\alpha_i) \neq y_i\}$. Consider the polynomial $E(x) = \prod_{i \in T} (x - \alpha_i)$, it is easy to see that E is an error locator polynomial for y ,and degree$(E) = t$. □

Though we have shown the existence of an error locator polynomial for a given input word y, it is not yet clear how to compute this polynomial efficiently. Note that once we know an error locator plolynomial E, then we can compute the required polynomial using the standard Lagrangian interpolation. Towards computing E, define a new polynomial $N(x)$ as follows:

$$N(x) \stackrel{\triangle}{=} P(x) * E(x), \Longrightarrow P(x) = \frac{N(x)}{E(x)}$$

Note that we are interested in some error locator $E(x)$, and do not know any of the polynomials $P(x)$, $E(x)$ and $N(x)$. Though computing each of the polynomials seems to be hard, we can compute them simultaneously!. In particular, we will compute N and E and hence P can be computed using the above relation. The idea is to view the coefficients of N and E as unknowns of a linear system of equations that we will write, and argue that the system is consistent and solvable.

Degree of $N(x) = (k-1) + t = k + t - 1 \leq k + t$. We are looking for the pair N, E and we want to ensure that they satisfy the following:

- $\forall i$, $E(\alpha_i) = 0$ iff $P(\alpha_i) \neq y_i$
- $N(x) = P(x) * E(x)$ with degree $< k + t$
- $\forall i$, $N(\alpha_i) = P(\alpha_i) * E(\alpha_i) = y_i * E(\alpha_i)$ (*got rid of P here!*).

Note that whenever $P(\alpha_i) \neq y_i$, $E(\alpha_i) = 0$ and hence we are justified in replacing $P(\alpha_i)$ with y_i in the above.

Algorithm 14.17 Berlekamp-Welch decoding algorithm

Input: $n, k, t \leq \frac{n-k-1}{2}$, $\{\alpha_1, \ldots, \alpha_n\} \subseteq \mathbb{F}$, $y = (y_1, \ldots, y_n) \in \mathbb{F}^n$.
Output: $P(x)$ with degree $P(x) \leq k-1$ such that $|\{i \mid p(\alpha_i) \neq y_i\}| = t$.

1: Compute two polynomials E, N such that
   - $\forall i$, $N(\alpha_i) = y_i * E(\alpha_i)$
   - degree of $E = t$
   - degree of $N \leq k + t$
2: Output $N(x)/E(x)$

[30] By abusing the notation, we let $\Delta(y, C) = \min_{c \in C} \Delta(y, c)$

Running time

LEMMA 14.4.3. *Time complexity of the Berlekamp-Welch algorithm is $O(n^3)$.*

Proof. We can write N and E as given below.

$$N = \sum_{j=0}^{k+t-1} N_j * x^j \text{ and } E = \sum_{j=0}^{t} E_j * x^j$$

$$\text{For all } 1 \leq i \leq n$$

$$N(\alpha_i) = y_i * E(\alpha_i)$$

$$\sum_{j=0}^{k+t-1} N_j * \alpha_i^j = y_i * \sum_{j=0}^{t} E_j * \alpha_i^j \quad \text{(A)}$$

$$t \leq \frac{n-k-1}{2} \Longrightarrow k + 2t + 1 < n + 2$$

The number of variables in the above set of linear equations is at most $n+1$, and the number of equations is n. We also need to add the constraint $E_t \neq 0$, which is not linear. Nevertheless, we can always assume that $E_t = 1$ (why?), and that adds one more equation to the above set of equations. If there exist such polynomials N, and E exist, then solving the system of equations above will give us one par (E, N) of polynomials. Then computing the fraction N/E is just a polynomial division, which can be done in $O(n)$ time. So the overall time required is the time required to solve the set of linear equations which is bounded by $O(n^3)$. □

Correctness : To complete the description of Berlekamp-Welch algorithm, we need to argue that the ratio N/E is always going to be the polynomial P independent of the solutions given by the algorithm for solving systems of linear equations. (Note that the error polynomial E is not unique, and hence we could get different solutions with different algorithms for solving linear equations.)

LEMMA 14.4.4. *If there are two pairs (N, E) and (N', E') which are the solutions of (A), then*

$$\frac{N(x)}{E(x)} = \frac{N'(x)}{E'(x)}, \text{ i.e. } N(x)E'(x) = N'(x)E(x).$$

Proof. Not that degree$(N(x)E'(x))$= degree$(N'(x)E(x)) \leq k + 2t - 1$. Now, $\forall i \in \{1, \ldots, n\}$,

$$\begin{aligned} y_i E(\alpha_i) &= N(\alpha_i); \quad \text{and} \\ N'(\alpha_i) &= y_i E'(\alpha_i) \\ \Rightarrow y_i E(\alpha_i) N'(\alpha_i) &= y_i N(\alpha_i) E'(\alpha_i). \end{aligned}$$

Now in fact, we show that $E(x)N'(x) = N(x)E'(x)$. This obvious if $y_i \neq 0, \forall i$. But if $y_i = 0$ for some i, then $N(y_i) = N'(y_i) = 0$, by the definition of N and N', and hence we get n points $\alpha_1, \ldots, \alpha_n$ where the polynomials $N(x)E'(x)$ and $N'(x)E(x)$ agree. However their degree is at most $k + 2t - 1 < n$, and hence the two polynomials should be the same. □

Exercise 14.4.5. Let C be an $[n,k,d]_q$ code. Let $y = (y_1, \ldots, y_n) \in (\mathbb{F}_q \cup \{?\})^n$ be a received word (where ? denotes an erasure) such that $y_i =?$ for at most $d-1$ values of i. Present an $O(n^3)$ time algorithm that outputs a codeword $c = (c_1, \ldots, c_n) \in C$ that agrees with y in all unerased positions (i.e., $c_i = y_i$ if $y_i \neq ?$) or states that no such c exists.(Recall that if such a c exists then it is unique.)

Curiosity 14.4.6 (Reed-Solomon Code for CDROMs). (This is a description from a lecture notes by Yehuda Lindell) CIRC stands for *Cross Interleaved Reed-Solomon code*; it was developed in 1979 by Phillips and Sony (the CD-ROM standard includes the error correcting code). The code provides a very high level of error correction and is the reason that even some scratched CDs typically keep working. This is a demonstration of the fact that the knowledge obtained in this part of the course is enough to understand real codes that are used.

Every disk contains a spiral track of length approximately 5 km. The track contains pits and lands (for representing bits). The width of the track is $0.6\mu m$ and the depth of a pit is $0.12\mu m$. The laser is shone onto the track and is reflected with a different strength if it is focused on a land or a pit, enabling the bit to be read. The errors are usually burst errors, meaning that they come in a continuous region.

Every audio sample is 16 bits long; in order to achieve stereo, two samples are taken for each time point. There are 44,100 samples per second and every sample pair is 4 bytes long. Therefore, every second of audio is made up of 176,400 bytes. Each sample in a pair is a vector in $\mathbb{F}_2^{16}$; each such vector is divided into two, and we view each half as an element of F_2^8. Two Reed-Solomon codes, denoted C_1 and C_2 are used in an interleaved fashion (this results in distributing large burst errors into small local errors).

14.5 Decoding Binary Reed-Muller Codes : Reed's Algorithm

Now we will present the decoding algorithm for binary Reed-Muller Codes. This variant of Reed-Muller codes was the one originally studied independently by Reed and Muller in the 1950s. The code is again based on using multivariate polynomials but over the field $\mathbb{F}_2$. In addition, the polynomials will also be multilinear - that is, each monomial has degree at most 1 in each variable participating in it. Hence each monomial corresponds to a subset of variables. Thus, the general form of the polynomial is:

$$p(x_1, x_2, \ldots x_r) = \sum_{\substack{S \subseteq [r] \\ |S| \leq t}} c_S \left(\prod_{i \in S} x_i \right)$$

Thus the encoding function interprets the message bits of $m \in \mathbb{F}_2^k$ as $(C_S)_{S \subseteq [r]}$ and evaluates the polynomial $p(x_1, x_2, \ldots x_r)$ on all 2^r substitutions in the set $\{0,1\}^r$. The dimension of the code is exactly the number of multilinear polynomials, which is exactly the number of subsets of $[r]$ of size at most t, which is $\sum_{i=1}^{t} \binom{r}{i}$. We use this in the following lemma:

LEMMA 14.5.1. *The binary Reed-Muller code is a* $\left[2^r, \sum_{i=1}^{t} \binom{r}{i}, 2^{r-t} \right]_2$ *code.*

Proof. We just need to show that the distance is exactly 2^{r-t}. Again, it suffices to argue about the

minimum weight of any non-zero codeword. We show that there is a non-zero codeword whose weight is at most 2^{r-t} and all non-zero codewords have weight at least 2^{r-t}.

The former can be observed by considering the subset $T = \{1, 2, \ldots, t\} \subseteq [r]$. Consider the message where c_S is non-zero exactly when $S = T$. The codword corresponding to this message produced by the above encoding will be non-zero and the polynomial will have exactly one monomial which is $p(x_1, x_2, \ldots x_r) = x_1 x_2 \ldots x_t$. The codeword will be non-zero for exactly 2^{r-t} elements in $\{0,1\}^r$ which sets $x_1 = x_2 = \ldots = x_t = 1$.

The latter can be argued by noting that any degree t polynomial must have a monomial of degree t and let S be the subset corresponding to that monomial such that $c_S = 1$. Hence,

$$p(x_1, x_2, \ldots x_r) = \prod_{i \in S} x_i + \sum_{\substack{T \subseteq [r] \\ |T| \leq t \\ T \neq S}} c_T \left(\prod_{i \in T} x_i \right)$$

Consider substituting values $\alpha = (\alpha_i)_{i \notin S} \in \mathbb{F}_2^{r-t}$ (That is, for the variables outside S in the subsets T). There are 2^{r-t} many ways of doing this. For each of these substitutions , the above polynomial gets restricted to a non-zero polynomial p' since the leading monomial cannot get cancelled by the substitutions. Hence each of them have a substitution in $\mathbb{F}_2^t$ where p' evaluates to non-zero. Thus for each partial assignment $\alpha \in \mathbb{F}_2^{r-t}$ there is an extension $\alpha' \in \mathbb{F}_2^r$ such that $p(\alpha') \neq 0$. Hence there are at least 2^{r-t} substitutions on which the polynomial evalutes to non-zero and hence the distance is at least 2^{r-t}. □

Reed's Decoding Algorithm: Let us recall the unique decoding problem in general. We are given a recieved message $y \in \{0,1\}^n$ with the guarantee that there is a $c \in C$ such that $d(y, c) \leq \frac{d-1}{2}$ - the decoding problem asks us to output m such that $E(m) = c$. Since binary Reed-Muller code is a $\left[2^r, \sum_{i=1}^t \binom{r}{i}, 2^{r-t}\right]_2$ code, we are given $y \in \mathbb{F}_2^{2^r}$ which is a binary string of length 2^r. We can think of this as the truth table of a function $f : \{0,1\}^r \to \{0,1\}$. Hence, we can restate this as follows : *Given $f \in \mathbb{F}_2^r \to \mathbb{F}_2$, with the guarantee that there is a polynomial $p(x_1, x_2, \ldots, x_r)$ with coefficients from $\mathbb{F}_2$ such that:*

$$|\{a \in \mathbb{F}_2^r \mid f(a) \neq p(a)\}| \leq 2^{r-t-1}$$

output the coefficients of the polynomial p. Notice that p will be unique by definition since the distance is 2^{r-t}.

Our target is to compute the coefficients of the polynomial p:

$$p(x_1, x_2, \ldots x_r) = \sum_{\substack{S \subseteq [r] \\ |S| \leq t}} c_S \left(\prod_{i \in S} x_i \right)$$

The following lemma will give us a direct computation method for the coefficients of the polynomial.

LEMMA 14.5.2. *For all $b \in \mathbb{F}_2^{r-t}$, and polynomial p of degree at most t, for any set $S \subseteq [t]$: $c_S = \sum_{\substack{a \in \mathbb{F}_2^r \\ a|_{\overline{S}} = b}} p(a)$*

To explain the expression on the RHS. Let us denote it to be $W(p,b)$. Thus, $W(p,b)$ is the sum of the polynomial evaluations on $a \in \{0,1\}^r$ which has b on indices outside the set S. We use the lemma for error correction, before proving it.
Suppose $f \in \mathbb{F}_2^r \to \mathbb{F}_2$ is given as the input. We know that f is not too far from the polynomial p that we want to compute (we want to compute the coefficients c_S of p for differnet sets S). Here is a first cut : how bad will be $W(f,b)$ as an estimate for the coefficient c_S for a subset S? But then, for which b? Intuitively, $W(f,b)$ cannot be the correct estimate c_S for all b's because somewhere the error that f contains, has to show up.

To formalise this : Let $B_b = \{a \in \mathbb{F}_2^r \mid a|_{\overline{S}} = b\}$ which we will call the *bag* corresponding to $b \in \mathbb{F}_2^{r-t}$. Indeed, for $b \neq b' \in \mathbb{F}_2^{r-t}$, $B_b \cap B_{b'} = \phi$ and they form a partition of $\mathbb{F}_2^r$. Let us call an $a \in \mathbb{F}_2^r$ to be *corrupt evluation point* if $p(a) \neq f(a)$. And a bag B_b to be a *corrupt bag* if $a \in B_b$ it contains a corrupt evaluation point a.

For a given S, if we knew an uncorrupted bag B_b, then we have that $\forall a \in B_b, f(a) = p(a)$, and hence we have $W(f,b) = W(f,b) = c_S$. Thus, c_S can be computed directly using f. But then, we do not know how to compute an uncorrupted bag, or even to check whether a given bag is uncorrupted for a given S, because we do not know the polynomial p (which is what we have to compute !).

Now comes the critical observation. We are given the guarantee that there are at most 2^{r-t-1} corrupt evaluation points in $\mathbb{F}_2^r$. Hence, in the worst case, they can all be distributed among the bags (one in each bag, in the worst case), thus corrupting at most 2^{r-t-1} bags. Even then, there will be more than $2^{r-t} - 2^{r-t-1}$ uncorrupted bags which is majority of the number of bags (which is 2^{r-t}).

Still, we do not know any explicit uncorrupted bag to compute c_S. But by the above arugment, we know majority of the bags will be uncorrupted. Thus, we can just run over all $b \in \mathbb{F}_2^{r-t}$ and for each of the bags, compute c_S and output the most frequent value (that is, majority). This gives the following algorithm:

Algorithm 14.18 Reed's Decoding Algorithm for Binary Reed-Muller Codes

---

Input: r, t and $f : \mathbb{F}_2^r \to \mathbb{F}_2$ and $S \subseteq [r]$
Output: Coefficient of the monomial $\prod_{i \in S} x_i$ in the multilinear polyonomial $p \in \mathbb{F}_2[x_1, x_2 \ldots x_r]$ with degree t such that

$$|\{a \in \mathbb{F}_2^r \mid p(a) \neq f(a)\}| < 2^{r-t-1}$$

1: $count \leftarrow 0$.
2: **for** `each` $b \in \mathbb{F}_2^{r-t}$ **do**
3: Compute $c_S = \sum_{\substack{a \in \mathbb{F}_2^r \\ a|_{\overline{S}} = b}} f(a)$.
4: **if** $(c_S = 0)$ **then** increment $count$.
5: **end for**
6: **if** $[count > 2^{r-t-1}]$ **then** output $c_S = 1$, **else** output $c_S = 0$.

Proof of Lemma 14.5.2: We will create a special notation for the monomial indexed by a subset S as $m_S(x) = \prod_{i \in S} x_i$. Viewing this as a function from $\mathbb{F}_2^r \to \mathbb{F}_2$, we observe two properties about

this function. Let $|S| = t$.

$\sum_{a \in \mathbb{F}^\ell} m_S(a) = 1$: Indeed, only one of the elements $a \in \mathbb{F}^\ell$ will make the monomial evaluate to 1, and everything else make it evaluate to 0. Hence the sum will be exactly 1.

$\forall T \subset S, \sum_{a \in \mathbb{F}^\ell} m_S(a) = 0$: To see this, let $i \in S \setminus T$ which exists because of the assumption. We split the sum into two sums based on the value of a_i.

$$\sum_{a \in \mathbb{F}^\ell} m_T(a) = \sum_{\substack{a \in \mathbb{F}^\ell \\ a_i = 0}} m_T(a) + \sum_{\substack{a \in \mathbb{F}^\ell \\ a_i = 1}} m_T(a) = 2 \sum_{\substack{a \in \mathbb{F}^\ell \\ a_i = 0}} m_S(a) = 0 (\text{over } \mathbb{F}_2)$$

where the second equality follows because $i \notin T$ and hence $m_T(a)$ does not depend on the value of a_i.

Using these two observations, now we can complete the proof of the Lemma 14.5.2. Fix an arbitrary $b \in \mathbb{F}_2^r$ and substitute it in $p(x)$ in order to get a restricted polynomial $p_b(x)$ with the variable set as S. We can write :

$$p_b(x) = c_S m_S(x) + \sum_{T \subset S} c_T m_T(x)$$

Using this notation:

$$\begin{aligned} \sum_{\substack{a \in \mathbb{F}_2^r \\ a|_{\overline{S}} = b}} p(a) = \sum_{a \in \mathbb{F}_2^t} p_b(a) &= c_S \sum_{a \in \mathbb{F}_2^t} m_S(a) + \sum_{a \in \mathbb{F}_2^t} \left(\sum_{T \subset S} c_T m_T(a) \right) \\ &= c_S \sum_{a \in \mathbb{F}_2^t} m_S(a) + \sum_{T \subset S} \left(c_T \sum_{a \in \mathbb{F}_2^t} m_T(a) \right) = c_S \quad \square \end{aligned}$$

More on Reed-Solomon Codes & Applications

Lecturer : Jayalal Sarma

WEEK
15

In the unique decoding regime, to ensure that there is a unique codeword within the distance of t, we need the error parameter $t < d/2$. What if $t = \lfloor \frac{d+1}{2} \rfloor$? In this case we know that there will be at least two code words in the hamming ball $B(y, t)$. What can we do about it? How about obtaining a list of codewords that are within a distance of at most t from the given erroneous message? This leads us to the definition of the List Decoding Problem which we discuss in this lecture.

15.1 List Decoding & Johnson's Bound

DEFINITION 15.1.1 (**List Decoding Problem**). *Given a code $C \subset \Sigma^n$, a word $y \in \Sigma^n$, and a parameter $0 < p < 1$, compute a list S of codewords in C that are within a distance of pn from y, i.e., output $S = B(y, pn) \cap C$.*

To talk about efficient algorithms for list decoding of a code C, it is also necessary to ensure that the set S is not too large. Towards this, we set up the notion of List Decodable Codes:

DEFINITION 15.1.2 (**List Decodable Codes**). *A code C is said to be (p, L) list decodable, if $\forall y \in \Sigma^n$, $|B(y, np) \cap C| \leq L$.*

Thus, by definition, any $(n, k, d)_2$ code is $(\frac{\delta}{2}, 1)$ list decodable because $\frac{\delta}{2}$ is the unique decodability radius. Beyond $\frac{\delta}{2}$ it depends on the combinatorial stucture of the packing of words in the code - it is fathomable that the number of codewords at distance $\frac{\delta}{2} + \epsilon$ suddenly shoots up to exponential. The boundary at which it happens is something that we need to calculate. Hence, it is not clear if (p, L) list decodable codes exist. Following lemma gives an answer.

LEMMA 15.1.3. *Let $|\Sigma| = q, 0 < p < 1 - 1/q$. There is a code C with rate $1 - H_q(p) - 1/L$ such that C is (p, L) list decodable, where $H_q(p)$ is the q-ary entropy function depending only on p, q.*

The above lemma can be proved by a probabilistic argument by considering random codes in Σ^n. We will skip the proof. Though it tells us the existence of list decodable codes, we would prefer to know methods to find if a given code is list decodable. The answer is given by the Johnson bound which shows a possible bound for p for which the ball of radius pn centered at any word in Σ^n contains more than one codeword interestingly at most $\mathsf{poly}(n)$ codewords.

THEOREM 15.1.4 (**Johnson Bound**). *Any $(n,k,d)_q$ code C is $(J_q(\delta), qdn)$ list decodable, where*

$$J_q(\delta) = \left(1 - \sqrt{1 - \frac{q}{q-1}\delta}\right)(1 - \frac{1}{q})$$

and $\delta = d/n$, the relative distance.

An immmeidate corollary that we would like to derive is how useful it is. Let us apply it for $q = 2$. It says, any binary linear code is linear sized list decodable if the number of errors made is at most $\frac{1}{2}\left(1 - \sqrt{1-2\delta}\right)$ (you can check that this number is greater than $\frac{1}{2}$ when δ is more than $\frac{1}{2}$).

One can ask the question whether the Johnson's bound is tight. The answer is yes - it can be shown (we dont provide the proof here) there exist linear codes with relative distance δ such that we can find Hamming ball of radius larger than $J_q(\delta)$ with super-polynomially many codewords.

We will now prove Johnson's Bound for binary linear codes.

THEOREM 15.1.5 (**Johnson Bound for Binary Codes**). *Let C be an $[n,k,d]_2$ code. Then every $y \in \{0,1\}^n$ and for $\frac{e}{n} < \frac{1}{2}\left(1 - \sqrt{1-2\frac{d}{n}}\right)$,*

$$|B(y,e) \cap C| \leq 2dn$$

Proof. The proof is by a double counting argument. Fix a $y \in \{0,1\}^n$. Let the ball $B(y,e) = B = \{c_1, c_2, \ldots c_\ell\}$. We need to prove that $\ell \leq 2dn$. We will do translation first. Consider $c_i' = c_i - y$. This gives us words $B' = \{c_1', c_2', \ldots, c_\ell'\}$ such that each has weight at most e and the pairwise distance between them still remains the same because $d(c_i', c_j') = d(c_i - y, c_j - y) = d(c_i, c_j) \leq d$. We will double count the sum of the pairwise distances in B'. Consider the sum:

$$S = \sum_{i<j} d(c_i', c_j')$$

One easy estimate for the summation is to use the distance bound for each pair to get a lower bound : $S \geq \binom{\ell}{2} d$. On one hand, let us consider the $n \times \ell$ matrix M which has the words in B' as columns. Let w_r be the weight of the r-th row of this matrix. Counting the contribution of the columns to the sum S, whenever i-th and j-th column chosen to find the distance is having different entries in the r-th row of them, it contributes 1 to the summation. The number of such pairs is $w_r(\ell - w_r)$. Thus we have, $S = \sum_{r=1}^{n} w_r(\ell - w_r)$. Let $w = \sum_r \frac{w_r}{\ell} = \frac{1}{\ell}\sum_r w_r$. Since $\sum_r w_r$ is counting the number of 1s, which is also the sum of the weights of the c_i's it is upper bounded by $e\ell$. Hence, $w \leq \frac{1}{\ell}(e\ell) \leq e$.

$$S = \sum_{r=1}^{n} w_r(\ell - w_r) = \sum_{r=1}^{n} \ell w_r - \sum_{r=1}^{n} w_r^2 \leq \ell^2 w - \sum_{r=1}^{n} w_r^2$$

To estimate the two terms, we will use Cauchy-Schwartz Inequality between the vectors, $(w_1, w_2, \ldots w_n)$ and $(\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n})$. This gives, $\left(\frac{\sum_{r=1}^{n} w_r}{n}\right)^2 \leq \frac{\sum_{r=1}^{n} w_r^2}{n}$ and implies,

$$S \leq \ell^2 w - \frac{\left(\sum_{r=1}^{n} w_r\right)^2}{n} \leq \ell^2 w - \frac{(\ell w)^2}{n} \leq \ell^2 \left(w - \frac{w^2}{n}\right)$$

Now we match it up with the lower bound:

$$\frac{d\ell(\ell-1)}{2} \leq \ell^2 \left(w - \frac{w^2}{n}\right))$$

and this gives:

$$\ell \leq \frac{dn}{dn - 2nw + 2w^2} \leq \frac{2dn}{(n-2w)^2 - n(n-2d)} \leq \frac{2dn}{(n-2e)^2 - n(n-2d)} \leq 2dn$$

The last inequality follows from the assumption on e which makes the denominator to be more than 1. □

There is also an alphabet-free version of Johnson bound, which we state for our information.

THEOREM 15.1.6 (**Johnson Bound - alphabet free version**). *Any $[n,k,d]_q$ code with distance d is $(\frac{e}{n}, qnd)$-list decodable where $e \leq n - \sqrt{n(n-d)}$.*

15.2 List-Decoding Algorithm for Reed-Solomon Codes

We now describe an algorithm for list decoding of Reed-Solomon codes. To know what to expect, let us apply the alphabet-free version of Johnson's bound. Since $d = n - k + 1$ for Reed-Solomon codes, this gives, that Reed-Solomon code has at most $O(n^3)$ codewords in any Ball of radius e where $e < n - \sqrt{n(n-(n-k+1))}$. Thus, Reed-Solomon code is (ρ, n^3)-list-decodable where $\rho < 1 - \sqrt{R}$. Do efficient algorithms exist which can list decode Reed-Solomon codes? We first formulate the problem with the specific details of Reed-Solomon codes.

PROBLEM 15.2.1 (**List Decoding Reed-Solomon codes**). *Given $(\alpha_1, y_1), \ldots, (\alpha_n, y_n) \in \mathbb{F}^2$ and $t \in \mathbb{N}$, output all polynomials $p(x)$ of degree at most $k-1$ such that*

$$|\{i \mid p(\alpha_i) = y_i\}| \geq t$$

Notice the change of notation. Rather than counting disagreements, we will count *agreement* between evaluations of p and the received vector y. The following remarkable result was proved in 1997, matching the Johnson bound for Reed-Solomon Codes.

THEOREM 15.2.2 (**Guruswami-Sudan(1997)**). *There is a polynomial time algorithm (runs in time $\mathsf{poly}(n)$) that can solve the list decoding problem given a guarantee that $t > \sqrt{nk}$.*

Earlier than this, Sudan had described an algorithm which list decodes Reed-Solomon codes given the guarantee that $t > 2\sqrt{nk}$. The list size in this case is $\sqrt{\frac{n}{k}}$ which is $\frac{1}{\sqrt{R}}$ for the error bound $n - \sqrt{2nk} = 1 - \sqrt{2R}$. The smaller the rate $R = \frac{k}{n}$ of the code, the list size is high. Intutively, in this case the codewords will be more tightly packed and the list of words within the radius itself is

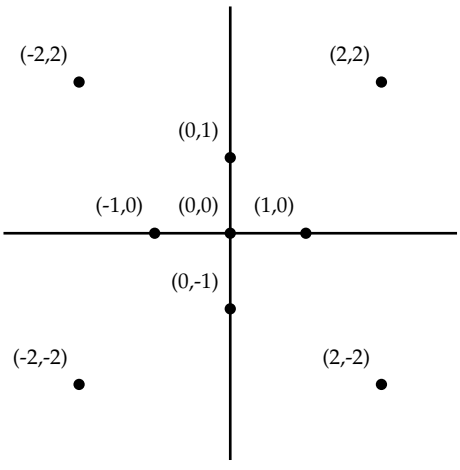
higher. We will prove an even weaker version of the theorem, to list decode when the agreement is at least $(k+1)\sqrt{n}+1$. In fact the same algorithm, with a tighter analysis will give the required bounds as well.

Extending Berlekamp-Welch Idea: To get to the general case of the algorithm, let us quickly recall Welch-Berlekamp Decoder for Reed-Solomon Codes. The decoding algorithm, given $y \in \mathbb{F}_q^n$ first finds a pair polynomials $(N(x), E(x))$ s.t. $N(\alpha_i)/E(\alpha_i) = y_i$ holds for all $i \in [n]$ provided $E(\alpha_i) \neq 0$. And then output $N(x)/E(x)$ as a polynomial. This can be thought of as an interpolation problem.

Another way of thinking about the above task is that of finding a rational function[31] $r(x) = \frac{f(x)}{g(x)}$ such that if $g(\alpha_i) \neq 0$, then $r(\alpha_i) = \frac{f(\alpha_i)}{g(\alpha_i)} = y_i$. And whenever $g(\alpha_i) = 0$, then $f(\alpha_i) = 0$ as well and in this case we will allow $r(\alpha_i)$ to take any value. Keeping this view, the Berlekamp-Welch aim can be reformulated as *Find a rational function $r(x)$ such that $r(\alpha_i) = y_i$ for all i whenever the denominator of the rational function evaluates to non-zero.* Indeed, thet theorem that we proved for Berlekamp-Welch decoding can be interpreted as : There exists a pair of polynomials (N, E) exists with $deg(N) \leq \frac{n+k}{2}$ and $deg(E) \leq \frac{n-k}{2}$ such that $\frac{N}{E}(\alpha_i) = y_i$ holds whenever $E(\alpha_i) \neq 0$.

High level Idea: Given a received word y, how do we output a set of polynomials which are "close" to y? One possibility is to output a product of those polynomials and then factorize it to get the individual polynomials. The above discussion gives the clue that we should be looking for more general algebraic objects to capture the data about the evaluation of the polynomial. It turns out that It turns out we can use bivariate polynomials - we will do a curve fitting through the i pairs (α_i, y_i) of points to get a bivariate polynomial $Q(x, y)$ and then factorise Q to get the polynomials of the form $y - p(x)$. Fortunately, factorization of bivariate polynomials can be done in time polynomial in the degree.

Example 1: We will take a more "curve fitting" view now. Consider the n input pairs as points in a two-dimentional plane. We will try to fit in a bivariate polynomial that passes through all the points and try to strike a relationship between the different $p_i's$ and the bivariate polynomial. Consider the nine points on the plane.



To obtain the polynomial in this case, define :

$$\begin{aligned} Q_1(x,y) &= x^2 + y^2 - 1 \\ Q_2(x,y) &= x + y \\ Q_3(x,y) &= x - y \end{aligned}$$

Now the bivariate polynomial passing through all the nine points is: $Q(x, y) = Q_1 Q_2 Q_3$. Indeed, there are many challenges 1) which all points will be together in which Q_i and 2) the bivariate polynomial passing through all the given points need not be unique.

[31] Rational function over a field $\mathbb{F}_q$ is the ratio of two polynomials defined over $\mathbb{F}$

Example 2: Given $\alpha_1, \ldots, \alpha_n \subseteq \mathbb{F}$, $y = (y_1, \ldots, y_n) \in \mathbb{F}^n$. Suppose there exist two degree $k-1$ polynomials $p_1(x), p_2(x)$ and $T \subset [n]$, $|T| = n/2$ such that

$$y_i = \begin{cases} p_1(\alpha_i) & \text{if } i \in T \\ p_2(\alpha_i) & \text{otherwise.} \end{cases}$$

The problem is to compute the polynomials p_1 and p_2. Observe that $\forall i, y_i = p_1(\alpha_i) \vee y_i = p_2(\alpha_i)$, and hence

$$\text{For all } i \in [n], \text{ we have : } (y_i - p_1(\alpha_i))(y_i - p_2(\alpha_i)) = 0$$

Now, if we define $Q(x, y) = (y_i - p_1(x))(y_i - p_2(y))$, then factorizing the polynomial $Q(x, y)$ would give us the answer. Since we do not yet know $p_1(x)$, and $p_2(x)$, we can try to find a polynomial Q from the data that $Q(\alpha_i, y_i) = 0$, $\forall i$, and then factorize Q. The algorithm can be described as follows.

1. Compute a polynomial $Q(x, y)$ of the form $y^2 + B(x)y + C(x)$, where B is a degree $k-1$ polynomial, and C is a degree $2(k-1)$ polynomial such that $Q(\alpha_i, y_i) = 0$, $1 \leq i \leq n$. This can be done by solving for the coefficients of $Q(x, y)$, by the linear equations

   $$y_i^2 + B(\alpha_i)y_i + C(\alpha_i) = 0 \ \ \forall i,$$

   with coefficients of B, and C as $3(k-1)$ unknowns. Solution exists if $n > 3(k-1)$.

2. Factorize $Q(x, y)$ into $(y - p_1(x))(y - p_2(x))$, and output $p_1(x)$, and $p_2(x)$.

As factorizing bi variate polynomials can be done in polynomial time, the above algorithm can be implemented in time $poly(n)$. The following lemma sproves the correctness of the algorithm:

LEMMA 15.2.3. *For any polynomial $Q(x, y)$ omputed in Step 1 of the algorithm above, we have, $(y - p_1(x))|Q(x, y)$, and $(y - p_2(x))|Q(x, y)$.*

Proof. We will prove that $(y - p_1(x))|Q(x, y)$, and the lemma follows by symmetry. Consider $Q(x, y)$ as a polynomial in y with univariate polynomials in x as coefficients, i.e.,

$$Q(x, y) = Q'(y) = Q_0 + Q_1 y + y^2.$$

We use the fact : if β is such that $Q'(\beta) = 0$, we have $(y - \beta)|Q'(y)$. Hence, it is sufficient to show that $Q'(p_1(x)) \equiv 0$. We will show this by degree constraints. If for some i, $y_i = p_1(\alpha_i)$, then $Q'(\alpha_i) = Q(\alpha_i, p_1(\alpha_i)) = 0$, and this is true for at least $n/2$ of the index i. But degree$(Q'(x)) \leq 2(k-1) < n/2$, and hence it must be that $Q'(p_1(x)) \equiv 0$, and hence $(y - p_1(x))|Q'(y)$ and hence $(y - p_1(x))|Q(x, y)$. □

General Form - Bivariate Interpolation Learning from the examples, given a set of pairs $(\alpha_i, y_i)_{i \in [n]}$ we would fit them in an algebraic curve in the plane, that is, we would construct a bi-variate polynomial $Q(x, y)$ s.t. $Q(\alpha_i, yi) = 0$ for all $i \in [n]$. We first show existance and construction of Q.

LEMMA 15.2.4 (**Bi-variate Interpolation**). *For every set of pairs $(\alpha_i, y_i)_{i\in[n]}$, there exists a non-zero bivariate polynomial $Q(x,y)$ with $\deg_x(Q), \deg_y(Q) \leq \sqrt{n}$ such that*

$$Q(\alpha_i, y_i) = 0 \text{ for all } i \in [n]$$

Proof. Let the polynomial $Q(x,y)$ be:

$$Q(x,y) = \sum_{i,j\leq\sqrt{n}} c_{ij} x^i y^j$$

Consider the c_{ij}s, the coefficients of $Q(x,y)$ as the unknowns and consider linear constraints among them represented by $Q(\alpha_i, y_i) = 0$ for all $i \in [n]$. Using the degree bound of $\sqrt{n}$ in each variable, the number of unknowns is $(\sqrt{n}+1)^2$. The number of constraints is n making it an underdetermined consistent system (since zero polynomial is always a solution) and hence there are many solutions. We can compute the non-trivial solution for $Q(x,y)$ by solving this linear system. □

Sudan's Algorithm: We are now ready to describe the algorithm:

Algorithm 15.19 : Sudan's List Decoding Algorithm for Reed-Solomon Codes

Input: $(\alpha_i, y_i)_{i\in[n]}$ and $t > (k+1)\sqrt{n}$

Output: A list of polynomials $p_j(x)$ of degree $\leq k$ such that $p_j(\alpha_i) = y_i$ for at least t of the $i \in [n]$.

1: Find $Q(x,y)$ with $\deg_x(Q), \deg_y(Q) \leq \sqrt{n}$.
2: Factorise $Q(x,y)$ to find an explicit list of factors of the form $y - p(x)$.
3: Output all those $p(x)$ such that $p(\alpha_i) = y_i$ for at least t values of $i \in [n]$.

Indeed, Step (1) in the algorithm can be done in $O(n^2)$ time since it is suffiicent to solve a system of linear equations for n variables. Step (2) can be done in $\mathsf{poly}(n)$ time by a known efficient algorithm for bivariate factorization. Step (3) can be done in polynomial time, if we prove that the number of polynomials produced in Step (2) is bounded by $\mathsf{poly}(n)$. Hence it suffices to provide an upper bound to the size of the list, the value of t and that completes the algorithm. How do we provide an upper bound to the list of polynomials. Indeed, the idea is that for each such polynomial, we argue that it contributes to the degree of $Q(x,y)$. Since there is an upper bound on the degree of $Q(x,y)$, we should have the upper bound on the list. The following lemma executes this plan.

LEMMA 15.2.5. *If $t > (\deg_x - 1) + (k-1)(\deg_y - 1)$, then $y - p(x) \mid Q(x,y)$.*

Proof. View $Q(x,y)$ as a polynomial in y with coefficient as polynomial expressions in x.

$$Q(y) = Q_0(x)y^0 + Q_1(x)y^1 + ... + Q_{\deg_y - 1}(x)y^{\deg_y - 1}$$

where $Q_i(x)$ is a polynomial in x with degree i. We use the simple algebraic property : $Q(\beta) = 0$ implies that $(y - \beta) \mid Q(y)$. Now, by our notation, $Q(p(x)) = Q(x, p(x))$. Hence, if $Q(p(x)) \equiv 0$ then $y - p(x)$ divides $Q(x,y)$ and we will be done.

Now we are left with the task of proving that $q(p(x)) \equiv 0$. This follows from degree considerations itself. Degree of the polynomial $Q(p(x))$ is atmost $d_x - 1 + (k-1)(d_y - 1)$. We show more than that many substitutions for x, which makes $Q(p(x))$ to be zero and hence conclude that $Q(p(x)) \equiv 0$. Indeed, we already have them, $Q(p(\alpha_i)) = Q(\alpha_i, p(\alpha_i)) = 0$ for atleast t indices and $t > d_x - 1 + (k-1)(d_y - 1)$. Thus we conclude that, $Q(p(x)) \equiv 0$ and hence $y - p(x)$ divides $Q(x, y)$. □

This immediately implies that the number of such factors can atmost be equal to $\deg_y$. Thus the list size is bounded by $\sqrt{n}$ and $t > k\sqrt{n-1}$. But we can do better. Let us write down the constraints that we have for the above arguments to work.

$$\begin{aligned} &\text{From Lemma 15.2.4 :} && d_x d_y > n \\ &\text{From Lemma 15.2.5 :} && t > d_x - 1 + (k-1)(d_y - 1) \end{aligned}$$

Thus, in the step 1 of the algorithm, we can look for Q with different degree constraints $\deg_x(Q)$ and $\deg_y(Q)$ such that the above constraints are satisfied. This gives a better list size if we choose $d_x = \sqrt{nk}$ and $d_y = \sqrt{\frac{n}{k}}$. The size of the list will be $\sqrt{\frac{n}{k}}$ and $t > (\sqrt{nk} - 1) + (k-1)(\sqrt{\frac{n}{k}} - 1)$. Hence if we insist that the agreement $t > 2\sqrt{nk}$ that satisfied the above constraint. By optimizing the choice of parameters, we can improve the required agreement constraint to $t > \sqrt{2nk}$. This is left as an exercise.

Curiosity 15.2.6 (Guruswami-Sudan Algorithm). To write the ideas required to improve the above agreement constraint to $t > \sqrt{nk}$ which matches (in relative to n) with the Johnson radius of $1 - \sqrt{R}$ for Reed-Solomon codes.

6: Jayalal says: Todo - Yet to write about Guruswami-Sudan Algorithm

Exercise 15.2.7 (Scrambled Reed-Solomon Codes). Let $\{\alpha_1, \ldots, \alpha_n\}$ be distinct elements of $\mathbb{F}_p$ used to define a Reed-Solomon code $C : \mathbb{F}_p^k \to \mathbb{F}_p^n$. Assume that $k < \frac{n}{6}$. For this problem, assume the fact (used in class) that for a bivariate polynomial $Q(x, y)$, we can find all its factors of the form $y - f(x)$.

(a) Suppose we sent two codewords corresponding to the polynomials P and P' (of degree $k-1$) but they got mixed up.Thus, we now have two lists $(\beta_1, \ldots, \beta_n)$ and $(\gamma_1, \ldots, \gamma_n)$ and we know for each $i \in [n]$:

$$\textbf{either } [P(\alpha_i) = \beta_i \text{ and } P'(\alpha_i) = \gamma_i] \textbf{ or } [P(\alpha_i) = \gamma_i \text{ and } P'(\alpha_i) = \beta_i]$$

Assuming that each co-ordinate is independently scrampled and we do not know which way each co-ordinate got scrambled, write down an algorithm to find out P and P'. [Hint: First find $P + P'$ and PP']

(b) Now, suppose that instead of getting both the values $P(\alpha_i)$ and $P'(\alpha_i)$ for each i, we only got one value β_i, such that for each i we either have $\beta_i = P(a_i)$ or $\beta_i = P'(a_i)$. Again, assume that each co-ordinate is independently scrampled and we do not know which way each co-ordinate

got scrambled. We are given the promise that:

$$\frac{n}{3} \leq |\{i \in [n] \mid \beta_i = P(\alpha_i)\}| \leq \frac{2n}{3} \quad \text{and} \quad \frac{n}{3} \leq \left|\{i \in [n] \mid \beta_i = P'(\alpha_i)\}\right| \leq \frac{2n}{3}$$

Design an algorithm to find out P and P'.

15.3 Algorithmic Applications of Reed-Solomon Codes

In this lecture we present two applications of Reed-Solomon codes in algorithm design. The algorithmic models are very different and one uses the structure of the code alone and the second one uses the efficient unique decoding and list decoding algorithms for Reed-Solomon Codes.

15.3.1 Communication Complexity

As mentioned above, this application does not use the decoding algorithms for Reed-Solomon Codes, but rather it uses the structure of the codewords. We will quickly introduce the communication complexity set up first.

Let $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ be a Boolean function. The communication complexity model has two parties Alice and Bob - each holding a string in $\{0,1\}^n$ each, say $x \in \{0,1\}^n$ with Alice and $y \in \{0,1\}^n$ with Bob, and they want compute the function value $f(x,y)$. That is, at the end of the algorithm (which in the context of communication complexity, are called *protocols*) both the parties should know the result $f(x,y)$. The resource of computation here is the number of bits that they exchange[32] expressed as a function of n. As in the case of algorithms, for a given function f, we would like to understand the number of bits exchanged, in the worst case over different x and y, by the best protocol to compute the function. This is denoted by $CC(f)$.

Indeed, a trivial way to compute $f(x,y)$ is the following protocol: Alice sends x across to Bob and Bob computes the function $f(x,y)$ and send the one bit answer back to Alice. This protocol takes $n+1$ bits of communication. This is independent of the function f. Hence, $CC(f) \leq n+1$. A general question is whether we can do better than $n+1$, even asymptotically.

Examples: Consider the function $\textsc{Par}(x,y) = \bigoplus_{i=1}^{n}(x_i \oplus y_i)$. This function has a 2-bit protocol which uses associativity of the $\oplus$ operator - Alice sends $b = \bigoplus_{i=1}^{n} x_i$ to Bob and Bob computes $\left(b \bigoplus \left(\oplus_{i=1}^{n} y_i\right)\right)$ and returns the bit b. Thus, $CC(\textsc{Par}) = 2$ and $\textsc{Par}$ is an easy function for this model of computation.

Let us take a second example. Consider $\textsc{Eq}(x,y) = \wedge_{i=1}^{n}(x_i = y_i)$. That is, to check if the two strings x and y are the same or not. A moments thought might convince you informally that one should probably expect this function to be harder in communication complexity model. Indeed, it is true - $CC(\textsc{Eq}) = n+1$. This requires proof though, and communication complexity offers quite a number of exciting proof techniques for showing upper and lower bounds. We leave that discussion aside since that is a digression at the moment.

We get to a randomized communication complexity model, where Alice and Bob also has a source of shared randomness (unbiased coin flipes). Indeed, the requirement is that over a good

[32] Alice and Bob have infinitel computational power an also knows each others computation strategy. What they do not know is each others n-bit input string.

fraction of outcomes of the random bits that they choose, the answer given by the protocol must be equal to $f(x,y)$. More formally, if $\mathcal{P}$ is the protocol:

$$\forall x,y \Pr_{r\in\{0,1\}^m}[\mathcal{P}(x,y,r) = f(x,y)] \geq \frac{1}{2} + \epsilon$$

Indeed, as in the case of randomized algorithms, one can imagine protocols which produces only one sided error, and protocols with two sided error (in which case, we want $0 < \epsilon \leq \frac{1}{2}$).

We attempt on a randomized algorithm for $\text{EQ}(x,y)$ function. A trivial attempt first. Suppose Alice and Bob chooses a random $i \in [n]$ (using their common randomness). Alice sends across x_i and Bob checks if $x_i = y_i$, if yes, Bob declares that the function value is 1 and otherwise 0. Let us analyse this randomized protocol. Indeed, if $x = y$, then the protocol never errs - no matter which i is chosen, $x_i = y_i$ will be satisfied. Hence it is a one-sided error randomized protocol. Suppose $x \neq y$. A moment of through will reveal the worst case setting where $x_i \neq y_i$ for exactly one $i \in [n]$ and the strings match at other indices. In this case, the probability of success is $\frac{1}{n}$. Note that we cannot afford to repeat the algorithm linear number of times since that increases the communication complexity to linear (which the trivial protocol also achieves anyway).

Using Reed-Solomon Codes to Design Randomized Protocols for $\text{EQ}(x,y)$: So a natural question is whether Alice and Bob can transform their inputs to some longer strings, where the above randomized algorithm (of choosing an index at random and checking for equality at the index for both their inputs) can be done. An intutive idea would be that, if there is one index in which the strings x and y differ, then there must be many differences in the new strings which gets caught by the randomized algorithm with noticable (constant) probability. Codes indeed have this natural property and as we will demonstrate below, Reed-Solomon codes, just does the job.

The idea is quite simple - Both Alice and Bob encodes their inputs as Reed Solomon Code words of a $[N, K, N-K+1]_q$ where $q \geq N$ and $K = n$. We will choose the blocklength N appropriately for the required error parameter. Let $\epsilon > 0$ be the target error. Thus, as a part of the encoding, Alice and Bob have polynomials P_x and P_y respectively of degree at most $K-1$ each. The codewords are evaluations of the polynomials on the elements in $\mathbb{F}_q$. Let $c_x, c_y \in \mathbb{F}_q^N$ be the codewords such that any $\alpha \in \mathbb{F}_q$ forms an index to the codeword, and $c_x[\alpha] = P_x(\alpha)$ and $c_y[\alpha] = P_y(\alpha)$. The protocol is same as described in the previous paragraph which we write in formal form below.

Algorithm 15.20 : Communication Protocol for $\text{EQ}(x,y)$ using Polynomials

Input: Alice has $x \in \{0,1\}^n$ and Bob has $y \in \{0,1\}^n$
Output: Both Alice and Bob must know $\text{EQ}(x,y)$.

1: Using the Reed-Solomon Code $[N, K, N-K+1]_q$, Alice and Bob independently computes $c_x = C(x)$ and $c_y = C(y)$ respectively (both in $\mathbb{F}_q^N$).
2: Choose an $\alpha \in \mathbb{F}_q$ uniformly at random from a common random source.
3: Alice sends $b = c_x[\alpha] \in \mathbb{F}_q$ to Bob. Bob checks if $b = c_y[\alpha]$. If yes, Bob declares $\text{EQ}(x,y)$ is 1, and 0 otherwise.

We argue the success probability. Suppose $x = y$, then $C(x) = C(y)$ as well. Hence no matter which α is chosen in step 2, Bob will end up declaring $\text{EQ}(x,y) = 1$. Suppose $x \neq y$, then, we

want to analyse the number of differnet $\alpha \in F_q$ such that $c_x[\alpha] = c_y[\alpha]$. Indeex, consider any such an α. We know that $P_x(\alpha) = P_y(\alpha)$. Hence, α is a root of the univariage polynomial $P_x - P_y$, which is of degree at most $n-1$. Hence, the number of such αs is bounded by $n-1$. Thus, the error probability,

$$\Pr_{\alpha \in \mathbb{F}_q}\left[c_x[\alpha] = c_y[\alpha]\right] = \Pr_{\alpha \in \mathbb{F}_q}\left[P_x(\alpha) = P_y(\alpha)\right] \leq \frac{n-1}{q} \leq \frac{n}{q}$$

Hence, if we have target error probability to be at most ϵ, then we have to choose $q = N \geq \frac{n}{\epsilon}$. The communication complexity is $O(\log \frac{n}{\epsilon})$. If ϵ is a constant, then we have an $O(\log n)$ cost protocol for $\text{EQ}(x, y)$. Compare this with $\Omega(n)$ lower bound that we mentioned in the beginning of this section. Thus, in the setting of communication complexity, randomization is provably more powerful and these protocols cannot be derandomized without incurring linear cost overhead.

One final remark about this applicatin is that it is a bit artificial since we did not use any property about the code such as distance and decoding algorithms. In the next section, we present a "real" application of codes.

Exercise 15.3.1 (Protocol using Hadamard Codes). Come up with a protocol for $\text{EQ}(x, y)$ function with cost $O(\log n)$ bit of communication, using Hadamard Codes.

15.3.2 Self-Correcting Algorithms for Permanent

The driving question for this section is the following possibility : Can we have a mechanism for tranforming our programs which may fail on small fraction of inputs (that is, the algorithm works correctly on most of the inputs) into programs that are usually correct into randomized programs that are always correct, with high probability. Notice that we want to do this without using any details of the program that we are tranforming and hence should be done in a blackbox way. Interestingly, this can be done for certain problems, in fact, in general for algorithms whose task is to evaluate implicit polynomials. We make this more precise as we go further.

Let p be a prime. We will consider the algorithmic problem of computing the determinant of an $n \times n$ matrix $A = \left(A_{i,j}\right)_{i,j=1}^{n} \in \mathbb{F}_p^{n \times n}$. Computing the determinant by highschool method involves recursively computing the cofactors and then adding up them. A quick analysis shows that this recursion follows the recurrence $T(n) = nT(n-1) + n$ which gives a runtime of $2^{O(n \log n)}$. A different formulat for the determinant which is derived by using the properties[33] of the determinant - is the Leibniz formula for determinant is

$$\det(A) = \sum_{\sigma \in S_n} \text{sign}(\sigma) \prod_{i=1}^{n} A_{i,\sigma(i)} \tag{15.34}$$

where S_n is the set of all permutations on n symbols, and sign is the sign function of a permutation, i.e $\text{sign}(\sigma) = -1$ if the number of inversions in σ is odd, and 1 otherwise.

Even Leibniz formula does not help in computing the determinant of a given $n \times n$ matrix efficiently. The algorithm which explcitly computes the summands in Leibniz formula will take time $O(n!) = 2^{O(n \log n)}$ time. This is impractically difficult for large n. Instead, the determinant can be evaluated in $O(n^3)$ operations by forming the LU decomposition of A as $A = LU$. In this

[33] such as exchanging two columns will change sign of the determinant, such as having two columns identical results in zero-value for determinant etc. See here for more details.

case, $\det A = (\det L)(\det U)$ and the determinants of the triangular matrices L and U are simply the products of their diagonal entries. The LU decomposition is computed by using Gaussian elimination.

A close cousin of the determinant is called the Permanent function of a matrix. The expression for this function is similar when we write in the Leibnis form - by dropping the sign function from the definition of determinant function. That is, for an $n \times n$ matrix,

$$\mathsf{perm}(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} A_{i,\sigma(i)}.$$

Indeed, given the matrix A, computing the permanent does not yield to an algorithm. An immeidate approach is to follow the above expression explicitly, but as observed this requires at least $2^{O(n \log n)}$ time. In the case of determinant, it had special alternating properties which makes the Gaussian elimination work and hence gave an $O(n^3)$ algorithm. But in the case of permanent, there is no useful property like that. In fact, it is conjectured that permanent cannot be computed in polynomial time. In his seminal paper, Valiant showed that computing perm_n is polynomial time equivalent to counting the number of satisfying assignment of a Boolean formula. Noting that even testing whether there is a satisfying assignment for a given formula or not is considered to be a difficult problem (NP-complete), and counting the number of satstying assignments is harder.

Characterstic of the Field Matters: We should remark about finite fields since that is what we are going to work with. Consider the case when $p = 2$. Over $\mathbb{F}_2$, notice that $+$ and $-$ operators yield the same result, and hence sign can be dropped from determinant. In other words, over $\mathbb{F}_2$, computing the determinant is same as computing the permanent of a matrix with entries from $\mathbb{F}_2$. Hence, the Gaussian elimination computes the permanent of the matrix itself over $\mathbb{F}_2$. Indeed, even the above reduction due to Valiant requires p to be large.

Randomized Algorithms for $\mathsf{perm}(A)$?: A natural question to ask is whether we can have randomized algorithms for computing the permanent of a matrix. To make the model precise, given a matrix A, the algorithm $\mathcal{R}$ can toss random bits $r \in \{0,1\}^m$, but should provide the guarantee:

$$\Pr_{r \in \{0,1\}^m} [\mathsf{perm}(A) = \mathcal{R}(A, r)] \geq \frac{1}{2} + \epsilon$$

It is worth noting that having efficient randomized algorithms for computing the permanent will immediately yield efficient randomized algorithms for testing satisfiability of a given formula ϕ. This is currently unknown, and is conjectured to be not true. Hence, *it is very unlikely that the task of computing permanent of a given matrix can be performed through an efficient randomized algorithm*. This last statement is what we rely on in this lecture.

Algorithms with Correct-on-Most-Inputs type guarantee: Note that the above lower bound for perm is as worst-case lower bound, and still does not eliminate the possibility of a polynomial time algorithm A such that A computes permanent correctly for a constant fraction of matrices $B \in F_p^{n \times n}$.

Applying Unique-Decoding Regime : Gimmel-Sudan Algorithm

Continuing from the previous discussion, we ask the following question: *Is there algorithm $\mathcal{A}$ such that $\mathcal{A}$ runs in time* $\mathsf{poly}(n)$ *and computes* $\mathsf{perm}(A)$ *for at least α fraction of input matrices $B \in \mathbb{F}_p^{n\times n}$, for a constant $\alpha > 0$?* We use the unique decoding regime of Reed-Solomon codes, in order to answer the above question as NO, by showing that if there is such an algorithm, then there is an efficient randomized algorithm which computes permanent on all matrices with high probability.

Random Tranlations of Matrices: We demonstrate the idea by presenting a simpler case when $\alpha = 1 - O(1/n)$. Let $p > n$. This one does not use codes.

THEOREM 15.3.2 (**Lipton**). *If $\mathcal{A}$ is an algorithm that computes* perm *for all but a $\frac{1}{3(n+1)}$ fraction of matrices in $\mathbb{F}_p^{n\times n}$, then there is a randomized polynomial time algorithm computing* perm *on all matrices $B \in \mathbb{F}_p^{n\times n}$, with probability at least $2/3$.*

Proof. Suppose we have an algorithm $\mathcal{A}$, that computes perm for all but a $\frac{1}{3(n+1)}$ fraction of matrices in $\mathbb{F}_p^{n\times n}$. Our aim is to design a randomized algorithm that computes perm on all input $B \in \mathbb{F}_p^{n\times n}$ matrices with probability at least $2/3$. For this, consider an input matrix $B \in \mathbb{F}_p^{n\times n}$, define the matrix

$$D(x) = B + xC$$

where C is a matrix chosen uniformly at random from $\mathbb{F}_p^{n\times n}$ and x is an indeterminate. The first observation is that since B is fixed as the input matrix, for any substitution of $t \in \mathbb{F}_p$, $D(t)$ is unformly distributed over the set of matrices in $\mathbb{F}_p^{n\times n}$. In particular, since $p > n$, for $i \in [n+1]$, $D(i)$ is also uniformly distributed in $\mathbb{F}_p^{n\times n}$.

The matrix D has variable x appearing in various entries. Consider $P(x) = \mathsf{perm}(D(x))$. Notice that $P(x)$ is a polynomial of degree at most n in x - since any permutation can multiply at most n appearances of the variable x in the expression in Equation 15.34. If we somehow have access to this polynomial $P(x)$, then $P(0) = \mathsf{perm}(D(0)) = \mathsf{perm}(B)$. Hence, $P(0)$ is the answer that we are looking for. Thus the strategy now is to look for ways to compute $P(x)$.

Let $Q(i) = \mathcal{A}(D(i))$, $i = 1, \ldots, n+1$. Note that $P(x)$ is a uni-variate polynomial of degree at most n, and $P(x) = \mathsf{perm}(B)$. By the property of algorithm $\mathcal{A}$, and since $D(i)$ is uniformly distributed in $\mathbb{F}_p^{n\times n}$, we have:

$$\forall i \in [n+1] \quad \Pr_{C\in\mathbb{F}_p^{n\times n}}[Q(i) \neq P(i)] \leq \frac{1}{3(n+1)} \tag{15.35}$$

$$\text{Hence, by union bound,} \quad \Pr_{C\in\mathbb{F}_p^{n\times n}}[\exists i \in [n+1],\ Q(i) \neq P(i)] \leq 1/3 \tag{15.36}$$

$$\text{Hence,} \quad \Pr_{C\in\mathbb{F}_p^{n\times n}}[\forall i \in [n+1],\ Q(i) = P(i)] \geq 2/3 \tag{15.37}$$

That is, with probability at least $2/3$ over the choice of $C \in \mathbb{F}_p^{n\times n}$, we have the evaluations of the polynomial P (of degree at most n) in $n+1$ points. Thus, we get $n+1$ points through which the polynomial $P(x)$ must pass through. Let them be $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$. We can directly

write the polynomial as:

$$P(x) = \sum_{i=0}^{n} y_i \left(\prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{(x - x_i)}{(x_j - x_i)} \right)$$

And, $P(0) = \mathsf{perm}(B)$ is the required output. In fact, this gives a direct formula for the permanent.

$$\mathsf{perm}(B) = \sum_{i=1}^{n+1} \mathcal{A}(D(i)) \left(\prod_{\substack{j=1 \\ j \neq i}}^{n+1} \frac{i}{(i - j)} \right)$$

where, all operations are in $\mathbb{F}_p$ and since $p \geq n + 1$, $[n + 1]$ is identified with the corresponding elements in $\mathbb{F}_p$ (in this case $\mathbb{Z}_p$). We describe the randomized algorithm for perm:

Algorithm 15.21 : Lipton's Randomized Algorithm for perm using $\mathcal{A}$

Input: $B \in \mathbb{F}_p^{n \times n}$. **Output:** $\mathsf{perm}(B)$.

1: Choose $C \in \mathbb{F}_p^{n \times n}$ uniformly at random.
2: Let $D(x) = B + xC$.
3: Run $\mathcal{A}$ on $D(1), \ldots, D(n + 1)$ to obtain values $y_1 = \mathcal{A}(D(1)), \ldots, y_{n+1} = \mathcal{A}(D(n + 1))$.
4: Interpolate to get a polynomial Q with $\deg(Q) \leq n$ such that $\forall i \in [n + 1], Q(i) = y_i$.
5: Output $Q(0)$.

By the argument above, $\Pr_{C \in \mathbb{F}_p^{n \times n}} [Q \equiv P] \geq 2/3$ and hence the correctness follows. □

Strengthening Lipton's Theorem beyond $\frac{1}{3n+1}$ error bound: From Lipton's theorem, we concluded that it is unlikely to have a deterministic algorithm computing permanent on all but $\frac{1}{3n+1}$ fraction of matrices in $\mathbb{F}_p^{n \times n}$. But then, can there be an efficient algorithm which does signifinantly worse than this - that say, computes correctly on all but $\frac{1}{4}$ fraction of matrices in $\mathbb{F}_p^{n \times n}$. In this section, we will use Reed-Solomon codes in order to achieve a similar reduction upto an error bound of $\frac{1}{2} - \epsilon$.

An instructive attempt: A natural attempt is ensure we have evaluations of $D(x)$ on more points via the algorithm $\mathcal{A}$. But how much more? Let us call this number N. We view $(P(1), \ldots, P(N))$ which are the correct evaluations as a *codeword* in an $[N, n, N - k + 1]$ Reed-Solomon code, where $p \geq N \geq n + 1$ and view the error made by the algorithm as the corruptions made by the channel. Given the above set of parameters, instead of using interpolation, we could use the Berlekamp-Welch decoding algorithm to get the polynomial $P(x)$ by correcting at most $(N - k + 1)/2$ errors in the received word $(Q(1), \ldots, Q(N))$. Thus we can allow $\mathcal{A}$ to err on at most $(N - k + 1)/2$ of the matrices in $\{D(i) \mid i = 1, \ldots, N\}$. But then if the $D(i)$s where independent, then situation would have been easy - it suffices if error probability of $\mathcal{A}$ is be at most $(N - k + 1)/2N$, then we can recover the polynomial $P(x)$ from the values $(Q(1), \ldots, Q(N))$ by applying Berlekamp-Welch decoding algorithm. Thus, setting $N = \frac{n}{2\epsilon}$ for $\epsilon > 0$ would have given the desired bounds.

But unfortunately, $D(i)$s are not independent. Hence, we ended up using union bound (in Equation 15.36) to deduce that with constant probability, all the evaluations are going to be correct. As discussed above, here the constraints are different: we cannot expect the algorithm to

succeed on all of the instances for any setting of the random bits. The best that we can hope for is to get the correct answers on αN of the input points. We need to establish such a concetration of the probability. For this, we need to design D with more indepednence. It turns out that pairwise independence would suffice, and apply Chebyshev's inequality to conclude that with high probability, the number of correct answers is indeed close to αN. [**?**] implemented exactly that.

THEOREM 15.3.3 ([**?**]). *For any $\epsilon > 0$, if $\mathcal{A}$ is an algorithm that computes* perm *for all but a $\left(\frac{1}{2} - \epsilon\right)$ fraction of matrices in $\mathbb{F}_p^{n\times n}$, then there is a randomized polynomial time algorithm computing* perm *on all matrices $B \in \mathbb{F}_p^{n\times n}$, with probability at least $2/3$.*

Proof. We construct (by careful sampling from $\mathbb{F}_p^{n\times n}$ a domain $D \subseteq \mathbb{F}_p^{n\times n}$) parametrized by a single variable x (that is, the points in the domain D are given by $\{D(x)|x \in \mathbb{F}_p\}$), such that D satisfies the following properties:

Property 1: The function $P(x) = \mathsf{perm}(D(x))$, is a univariate polynomial of degree at most $2n$ in x. The end-result that we want will be $P(0)$. So our aim will be to compute the polynomial $P(x)$ in explicit form.

Property 2: With high probability, $\mathcal{A}$ computes perm on approximately the same fraction of inputs from the domain D as from the domain $\mathbb{F}_p^{n\times n}$.

Notice that the two properties are contradictory - first one requires the domain D to be *nicely structured* while the second one requires that sampling from S should look like *random sample* of $\mathbb{F}_p^{n\times n}$. It is an interesting puzzle whether we can achieve both simultaneously. It turns out that if we choose D to be *pairwise independent samples* it achieves best of both worlds. This is the main idea proposed [**?**].

Construction: Let $B \in \mathbb{F}_p^{n\times n}$ be the input matrix. Let $S, T \in \mathbb{F}_p^{n\times n}$ chosen uniformly at random and define :

$$D(x) \stackrel{\text{def}}{=} \{Sx^2 + Tx + B\}$$

As in the case of Lipton's theorem, define $P(x) = \mathsf{perm}(D(x))$. For a given S and T, define the family as:

$$\mathcal{D}_{S,T} = \{D(x) \mid x \in \mathbb{F}_p\}$$

We argue the properties required. To see Property 1 $P(x)$ is, by definition, a polynomial in degree at most $2n$. Right away, we have that $P(0) = \mathsf{perm}(B)$. To see, Property 2, we claim that for any substitution i for x, $D(i)$ is distributed in $\mathbb{F}_p^{n\times n}$ uniformly, over the choice of matrices S and T. We need to conclude about the fraction of inputs from the domain D. We are aiming to use a Reed-solomon code of parameters $[N, K, N - K + 1]_p$ over the field $\mathbb{F}_p$. Choosing $K = 2n$, and N will be fixed later - we need to get at least $\frac{N-2n+1}{2}$ co-ordinates in the codeword right by the algorithm $\mathcal{A}$ to apply this framework. The following lemma achieves that with appropriate choice of parameters.

LEMMA 15.3.4. *Consider N elements from the set $D_{S,T}$ above:*

$$\Pr\left[\begin{array}{c}\mathcal{A} \textit{ computes } \mathsf{perm} \textit{ correctly on at least} \\ \left(N\left(\frac{1}{2}+\epsilon\right) - c\sqrt{N}\right) \textit{ samples}\end{array}\right] \geq 1 - \frac{1}{c^2}$$

To apply the lemma, let e be the number of elements in the sample set S of N samples in D, in which the algorithm $\mathcal{A}$ makes an error. If we choose the number of samples to be $N = (\frac{c}{\epsilon} + n)^2$, then with probability at least $1 - \frac{1}{c^2}$, we have that

$$e \leq \left(\frac{c}{\epsilon} + n\right)^2 \left(\frac{1}{2} - \epsilon\right) + c\left(\frac{c}{\epsilon} + n\right) \leq \frac{1}{2}\left(\frac{c}{\epsilon} + n\right)^2 - \left(\frac{c}{\epsilon} + n\right)\epsilon n \tag{15.38}$$

$$\leq \frac{N}{2} - cn - \epsilon n^2 < \frac{N - 2n + 1}{2} \tag{15.39}$$

The last inequality holds for large enough n. Hence, we can apply Berlekamp-Welch decoding algorithm directly to get the polynomial P and evaluate $P(0)$ to get the $\mathsf{perm}(B)$ correctly. Thus, $c \geq 2$ will achieve a success probability of $\frac{2}{3}$ and the algorithm runs in time $\mathsf{poly}(N)$, which is $\mathsf{poly}(n, \frac{1}{\epsilon})$.

Proof of Lemma 15.3.4. For any fixed i, $D(i)$ is uniformly distributed over $\mathbb{F}_p^{n \times n}$ (Prove this as an exercise !). We prove the following indepedence claim.

CLAIM 15.3.5. *The family $\mathcal{D} = \{D(x) \mid x \in \mathbb{F}_p\}$ is pairwise independent.*

Proof. Recall the definition of $D(x) \stackrel{\text{def}}{=} Sx^2 + Tx + B$ where S and T are chosen uniformly at random from $\mathbb{F}_p^{n \times n}$. To show pairwise independence, For any two matrices $S', T' \in \mathbb{F}_p^{n \times n}$, and $a, b \in \mathbb{F}_p$, we need to prove that:

$$\Pr_{S,T}\left[(D(a) = S') \wedge (D(b) = T')\right] = \frac{1}{p^{2n^2}}$$

This follows by observing that for each entry of the matrix (i, j), there is exactly one degree 2 polynomial $p_{ij}(x)$ such that $p_{ij}(0) = B_{ij}$ and $p_{ij}(a) = S_{ij}$ and $p_{ij} = T_{ij}$ (since degree 2 polynomial is uniquely determined by three evaluations). □

Now the lemma follows from the claim below about a tail inequality for pairwise independent random variables. A careful application of Chebychev inequality (using the fact that variance of the sum is the sum of variances, when the random variables are pairwise independent) gives:

LEMMA 15.3.6. *If S is a set of N samples from the pairwise independent space $D \subseteq \mathbb{F}_p^{n \times n}$. For any $I : U \to \{0, 1\}$. Then for any positive constant c:*

$$\Pr\left[|\mathbb{E}_{x \in U} I(x) - \mathbb{E}_{x \in D} I(x)| \geq \frac{c}{\sqrt{N}}\right] \leq \frac{1}{c^2}$$

To apply the lemma, we define $I(x)$ to be the indicator random variable of when the algorithm $\mathcal{A}$ computes $\mathsf{perm}(D(x))$ correctly. Indeed, $\mathbb{E}_{x \in U} I(x) = \frac{1}{2} + \epsilon$ and the above lemma estimates the tail of the distribution using Chebychev inequality. □

□

Curiosity 15.3.7 (Reselient Algorithms for Computing Polynomials). The framework that we discussed is applicable to the problem of correcting programs that compute multivariate polynomials over large finite fields and not just computing the permanent polynomial. This gives an

efficient procedure to transform any program that computes a multivariate polynomial $P(x)$ correctly on a $1/2+\epsilon$ fraction of its inputs ($\epsilon > 0$) into a randomized program that computes $P(x)$ correctly on every input with constant probability. In fact, this is the result proved by **[?]** and the argument presented in this lecture is a corollary of their proof.

Applying List-Decoding Regime : Cai-Pavan-Sivakumar Algorithm

Note that the **[?]** breaks down if the original algorithm $\mathcal{A}$ makes at least $1/2+\epsilon$ fraction of errors. However, by applying list decoding algorithms we can improve this - that is, $\mathcal{A}$ can be allowed to err on at most $1-\frac{1}{n^c}$ where c is a constant.

THEOREM 15.3.8 (**Cai-Pavan-Sivakumar [?]**). *Let $c > 0$ be a constant. Suppose there is a deterministic polynomial time algorithm $\mathcal{A}$ computing* perm *over $\mathbb{F}_p$ for all n, and $p \geq 9n^{2c+2}$ correctly on at least $1/n^c$ fraction of the inputs from $\mathbb{F}_p^{n\times n}$, then there is a randomized polynomial time algorithm computing* perm *on all matrices $B \in \mathbb{F}_p^{n\times n}$, with probability at least $2/3$.*

Proof. For the ease of presentation, we will also assume an upper bound on $p \leq n^{c'}$ for a constant $c' > c$. Now we get on to the proof. Let $\mathcal{A}$ be the algorithm as mentioned in the premise of the lemma, computing permanent correctly for at least $\alpha \geq 1/n^c$ fraction of the matrices in $\mathbb{F}_p^{n\times n}$, i.e.,

$$\Pr_{B\in\mathbb{F}_p^{n\times n}}[\mathcal{A}(B) = \mathsf{perm}(B)] \geq 1/n^c.$$

Consider $B \in \mathbb{F}_p^{n\times n}$. Let $B_{1,1}, B_{2,1}, \ldots, B_{n,1}$ where $B_{i,1}$ is the $(n-1)\times(n-1)$ matrix obtained by deleting ith row and 1st column of B. Note that $\mathsf{perm}(B) = \sum_{i=1}^{n} b_{i,1}\mathsf{perm}(B_{i,1})$, where $B = (b_{i,j})_{i,j=1}^{n}$. Let $\delta_i(x)$ denote the delta function for $i \in [n]$ i.e.,

$$\delta_i(x) = \begin{cases} 1 & \text{if } i = x, \\ 0 & \text{if } i \neq x. \end{cases}$$

Note that $\delta_i(x)$ is a degree n polynomial. Let

$$\alpha(x) = \prod_{i=1}^{n}(x-i),$$

and

$$D(x) = \sum_{i=1}^{n}\delta_i(x)B_{i,1} + \alpha(x)(S+xT)$$

where $S, T \in \mathbb{F}_p^{(n-1)\times(n-1)}$ chosen uniformly at random. Define $P(x) := \mathsf{perm}(D(x))$, then $P(x)$ is a degree $(n+1)(n-1) = n^2-1 < n^2$ polynomial. Note that if we knew $P(1), \ldots, P(n)$ we can determine permanent of B as:

$$\mathsf{perm}(B) = \sum_{i=1}^{n} b_{i1}P(i)$$

Thus, our aim now is to compute the polynomial $P(x)$ as in the earlier case. We need to ask for

evaluations of $P(x)$ by using $\mathcal{A}$ on $D(x)$ for different substitutions for x other than $\{1, 2, \ldots n\}$. Let

$$D = \{D(a) \mid a = n+1, \ldots, p\}.$$

Suppose the matrices S, T are chosen are uniformly and independently at random from $\mathbb{F}_p^{(n-1)\times(n-1)}$, then $D(i)$ is distributed uniformly from $\mathbb{F}_p^{(n-1)\times(n-1)}$. Moreover set D is a set of pairwise independent family of matrices in $\mathbb{F}_p^{(n-1)\times(n-1)}$ over the choice of S and T. Using this, the following lemma can be proved by applying the Chebyshev's inequality. We will defer the proof of the Lemma to the end and proceed with the proof of the Theorem.

LEMMA 15.3.9. *With probability at least* $1 - \frac{1}{(p-n)\alpha^2}$, *algorithm* $\mathcal{A}$ *correctly computes* perm *on at least* $\frac{\alpha}{2}$ *fraction of the matrices in* D.

We use the lemma with the following modelling of the situation. Let us define the graph of a polynomial f is needed,

$$G_f := \{(i, f(i)) \mid i = 1, \ldots, p\}.$$

Let $S = \{(i, \mathcal{A}(D(i)) \mid i = n+1, \ldots, p\}$, then by Lemma 15.3.9 we have $|S \cap G_P| \geq \frac{\alpha}{2}(p-n)$ with probability at least $1 - \frac{1}{(p-n)\alpha^2}$. Now that idea is to obtain a list T of polynomials of degree less than n^2 such that $|S \cap G_f| \geq \frac{\alpha}{2}(p-n)$ applying Sudan's list decoding algorithm with input $\{(i, A(D(i))) \mid i = n+1, \ldots, p\}$. However we need an upper bound on the size of such a list:

LEMMA 15.3.10 (**bound on the size of the list**). *There are at most* $\frac{3}{\alpha}$ *polynomials* f *such that*

$$|S \cap G_f| \geq \frac{\alpha}{2}(p-n)$$

Assuming the Lemma, we complete proof of Theorem as follows. Now, apply $\mathcal{A}$ to $D(i)$, for $i = n+1, \ldots, p$, then we know that $(\mathcal{A}(D(n+1)), \ldots, \mathcal{A}(D(p)))$ agrees with $(P(n+1), \ldots, P(p))$ on at least $\frac{(p-n)}{2n^c}$ of the co-ordinates with probability $1 - \frac{1}{(p-n)\alpha^2}$. Notice that,

$$\frac{(p-n)}{2n^c} > \sqrt{2(p-n)n^2} \text{ since } p \geq 9n^{2c+2}$$

Thus, the number of polynomials f of degree at most $n^2 - 1$ such that $(A(D(n+1)), \ldots, A(D(p)))$ and $(f(n+1), \ldots, f(p)))$ agree on at least $\sqrt{2(p-n)n^2} = \sqrt{2Nk}$ coordinates is at most $\frac{3}{\alpha} \leq 3n^c$, where $N = p - n, k = n^2$. Hence, the word $(P(n+1), \ldots, P(p))$ can be seen as a codeword in an $[L, k, L-k+1]_p$ Reed Solomon code. Thus, we can apply Sudan's list decoding algorithm in the input $\{(i, A(D(i)) \mid i = n+1, \ldots, p\}$, with $t = \frac{(p-n)}{2n^c} > \sqrt{2Lk}$. Let T be the list of polynomials of degree at most $k - 1 = n^2 - 1$ output by Sudan's algorithm. We have $|T| \leq \frac{3}{\alpha} \leq 3n^c$. Moreover, we know that T contains the polynomial $P(x)$.

Identifying the Correct Polynomial from the codword list T: All we need to do now is to separate $P(x)$ from the rest of the polynomials in T. As any two polynomials in T can have a common value on at most $n^2 - 1$ points in $\mathbb{F}_p$, there are at most $9n^{2c}(n^2-1)$ points in $\mathbb{F}_p$ such that more than two polynomials in T agree on each of those points. Since $p \geq 9n^{2c+2}$, there is a $v \in \mathbb{F}_p$ such that $f(v) \neq g(v) \ \ \forall f \neq g \in T$. By a Brute force search and evaluation of polyomials in the list on

each element in $\mathbb{F}_p$, we will find out this $v \in \mathbb{F}_p$ explicitly[34].

Now, if we have access to $\mathsf{perm}(D(v))$, then $P(x)$ can be isolated from T, and in turn we can compute $\mathsf{perm}(M)$. The only thing that remains to be done is computing $\mathsf{perm}(D(v))$. Note that $D(v)$ is an $(n-1) \times (n-1)$ matrix, and hence we can recursively apply the whole process with $D(v)$ as the starting matrix instead of M.

$$\begin{aligned}
\Pr\ [\ \text{Algorithm Errs}\ ] &\leq \Pr\ [\exists i :\ \text{Algorithm Errs at Recursive Level } i\ ] \\
&\leq \sum_{i=1}^{n} \Pr\ [\ \text{Algorithm errs at level } i\ ] \\
&\leq \sum_{i=1}^{n} \frac{1}{(p-n)\alpha^2} \leq \frac{n}{(p-n)\alpha^2} \leq \frac{n^{c+1}}{9n^{2c+2}} \leq \frac{1}{9n^{c+1}}
\end{aligned}$$

Summarizing, the randomized algorithm for perm can be described as follows:

Algorithm 15.22 : Cai-Pavan-Sivakumar's[**?**] Randomized Algorithm for perm using a given algorithm $\mathcal{A}$ such that for a constant c : $\Pr_{B \in \mathbb{F}_p^{n \times n}}[\mathcal{A}(B) = \mathsf{perm}(B)] = \alpha \geq 1/n^c$.

Input: $B \in \mathbb{F}_p^{n \times n}$.
Output: $\mathsf{perm}(B)$.

1: Define $D(x) \stackrel{\text{def}}{=} \sum_{i=1}^{n} \delta_i(x) M_{i,1} + \alpha(x)(S + xT)$ choosing $S, T \in \mathbb{F}_p^{(n-1)\times(n-1)}$ uniformly and independently at random.
2: Let $Q(i) := A(D(i)), i = n+1, \ldots, p$.
3: Apply Sudan's list decoding algorithm with input $\{(i, Q(i)) \mid i = n+1, \ldots, p\}$, $t = \frac{(p-n)}{2n^c}$. Let T be the list of polynomials of degree at most $n^2 - 1$ computed by Sudan's algorithm.
4: Compute $v \in \mathbb{F}_p$ such that $\forall\ f \neq g \in T, f(a) \neq g(a)$. Since there is such an a, this can be done by a brute force search.
5: Recursively compute the value $b := \mathsf{perm}(D(v))$. (Note that $D(v)$ is an $(n-1) \times (n-1)$ matrix, hence the dimension of the matrix goes down by one at each level of recursion)
6: Find an $f \in T$ such that $f(v) = b$ (i.e., f has to be P with high probability, why?).
7: Output $\sum_{i=1}^{n} b_{i,1} f(i)$.

To complete the proof, we prove Lemma 15.3.9 and Lemma 15.3.10.

Proof of Lemma 15.3.9. For $i = n+1, \ldots, p$, let Z_i as 1 if A succeeds on $D(i)$ and 0 otherwise. Notice that $\mathbb{E}[Z_i] = \alpha$. Define the average random variable:

$$Z = \frac{1}{p-n} \sum_{i=n+1}^{p} Z_i$$

Let us calculate $\mathbb{E}[Z]$. By linearity, this is $\frac{1}{(p-n)} \sum_{i=n+1}^{p} \mathbb{E}[Z_i]$. This gives, $\mathbb{E}[Z] = \alpha$. We apply

[34] This is where we use the upper bound of p in terms of $\mathsf{poly}(n)$.

Chebychev inequality:

$$\begin{aligned}\Pr\left[Z \leq \frac{\alpha}{2}\right] &= \Pr\left[Z - E[Z] \leq -\frac{\alpha}{2}\right] \\ &\leq \Pr\left[|Z - E[Z]| \geq \frac{\alpha}{2}\right] \leq \frac{\mathsf{Var}[Z]}{\left(\frac{\alpha}{2}\right)^2}\end{aligned}$$

Since the random variables $Z_{n+1}, \ldots, Z_p$ are pairwise independent, $\mathsf{Var}[Z] = \frac{1}{(p-n)^2}\sum_{i=n+1}^{p}\mathsf{Var}[Z_i] \leq \frac{p-n}{4}$ by using the trivial upper bound for variance[35] Thus we have:

$$\Pr\left[Z \leq \frac{\alpha}{2}\right] \leq \frac{1}{(p-n)\alpha^2}$$

□

Proof of Lemma 15.3.10. Suppose there are more than $\frac{3}{\alpha}$ polynomials f polynomials of degree$<$ n^2 such that $|S \cap G_f| \geq \frac{\alpha}{2}(p-n)$. Let T be the set of $\lceil\frac{3}{\alpha}\rceil$ many of them. Then we have

$$\begin{aligned}p - n &\geq |\cup_{f\in T} S \cap G_f| \geq \sum_{f\in T}|S \cap G_f| - \sum_{f\neq g\in T}|S \cap G_f \cap G_g| \text{ by inclusion-exclusion principle} \\ &= |T|\frac{\alpha}{2}(p-n) - \binom{|T|}{2}(n^2-1); \text{ since } f, g \text{ are of degree at most } n^2-1, |G_f \cap G_g| \leq n^2 - 1. \\ &= \frac{|T|}{2}\left(\alpha(p-n) - (|T|-1)n^2\right)\end{aligned}$$

Since $|T| = \lceil\frac{3}{\alpha}\rceil$, then $|T| < \frac{3}{\alpha} + 1$, and $|T| \geq \frac{3}{\alpha}$, then substituting these to the above equation appropriately,

$$\begin{aligned}p - n &> \frac{3}{2\alpha}\left(\alpha(p-n) - \frac{3}{\alpha}n^2\right) \\ &= \frac{3}{2}(p-n) - \frac{9n^2}{2\alpha^2} = (p-n) + \frac{1}{2}\left((p-n) - \frac{9n^2}{\alpha^2}\right)\end{aligned}$$

If we show that the second term is positive, we have a contradiction. This is true, because $p > 9n^{9c+2}$ and hence $p - n > \frac{9n^2}{\alpha^2}$. Thus we get a contridiction and hence there can be strictly less than $\frac{3}{\alpha}$ many polynomials f polynomials of degree$<$ n^2 such that $|S \cap G_f| \geq \frac{\alpha}{2}(p-n)$. and this conludes the proof of Lemma 15.3.10. □

This concludes the presentation of the Cai-Pavan-Sivakumar Algortihm. □

[35] Any 0-1 random variable has variance at most $\frac{1}{4}$. This is consequence of a more general theorem called *Popoviciu's inequality*. Let the random variable be X and $a \leq X \leq b$, then $\mathsf{Var}[X] \leq \frac{(b-a)^2}{4}$. A quick proof when $a = 0$, $\mathbb{E}[X^2] = \mathbb{E}[XX] \leq \mathbb{E}[bX]$. If $z = \mathbb{E}[X]$. We have $\mathsf{Var}(X) = \mathbb{E}[X^2] - (\mathbb{E}[X])^2$. $\mathsf{Var}(X) \leq cz - z^2$. This is maximised at $z = \frac{b}{2}$ and the maximum value is at most $\frac{b^2}{4}$.

Constructing Codes from Expanders

Lecturer : Jayalal Sarma

WEEK
16

In this lecture we will show how to construct codes from explicit expanders that we have constructed earlier in the course. These codes belong to a general family where we visualize the code as a graph G where the value of the i-th bit of a codeword corresponds to the value associated with the i-th vertex or the i-th edge of the graph. The distance of the code will then by given by combinatorial properties of the graph. The advantages of many of these codes will be that they have encoding and decoding algorithms that run in linear time.

The first graph-theoretic codes were proposed in 1963 by Gallager. In 1984, Tanner rediscovered and generalized graph-theoretic codes, replacing the parity-check constraint with more general linear codes, and also proposed the idea of using expander graphs. The idea was again rediscovered by Sipser and Spielman, who put together error-correcting codes and expander graphs.

We will work with bipartite expaners. That is, *a graph $G(U, V, E)$ is said to be a (n, m, d, α, β)-expander if, $|U| = n, |V| = m$, it is d-regular, and every $S \subseteq U$ such that $|S| = \alpha|V|$, the number of new neighbors $|N(S)| \geq \beta d|S|$*. We will be using D for the degree from now on to avoid confusion with the minimum distance of a code.

16.1 Expander Codes

Let $G(U, V, E)$ be a bipartite regular expander graph such that every vertex in U has exactly D neighbors in V, and $D > 2$. And let G be an expander with $\beta > \frac{3}{4}$. These parameters are chosen such that the following property is true : every subset in U which is at most $\alpha|U|$ size has many *unique neighbors*. Given $S \subseteq U$, we will say that $v \in V$ is a unique neighbor of S if there is exactly one vertex $u \in S$ such that $(u, v) \in E$. We write this as the following lemma.

LEMMA 16.1.1. *If $G(U, V, E)$ is a (n, m, D, α, β) bipartite expander where $D > 2$ and $\beta > \frac{3}{4}$, then every $S \subseteq U$, with $|S| \leq \alpha n$ has more than $\frac{D}{2}|S|$ unique neighbors.*

Proof. Let S be as in the statement. Let r be the number of unique neighbors of S. Then the number of edges going out from S is at most $D|S|$ and it produces $\beta D|S|$ many neighbors. But since r of them are unique neighbors of S, we have:

$$D|S| \geq r + 2(\beta D|S| - r)$$

And this implies that $r \geq (2\beta D - D)|S| > \frac{D}{2}|S|$. □

Construction: Define the following $m \times n$ matrix from the graph, $H[i,j] = 1$ if the edge $(j,i) \in E$. Now consider the code whose parity check matrix is the matrix H. That is, define the code as:

$$C = \{c \in \mathbb{F}_2^n \mid Hc = 0\}$$

There is a very natural combinatorial way to look at the code word with respect to the graph by using the above definition. For this, let us create a parity bit for vertices $v \in V$, $P_v = \bigoplus_{i \in N(v)} c_i$. A word $c \in \mathbb{F}_2^n$ is a codeword if and only if for every vertex $v \in V$, $P_v = 0$. That is, given a word $c \in \mathbb{F}_2^n$, it is a codeword if and only if for every vertex $v \in V$, the parity of the bits in y whose index is a neighbor of v in the graph G is zero.

Notice that the partiy check matix is the (transpose of) the adjacency matrix of a constant degree expander graph which by definition has only linear number of edges. Hence the parity check matrix has only few non-zero entries. Such codes, where the parity check matrix is sparse (that is it has only $\mathcal{O}(n)$ nonzero entries as opposed to $\mathcal{O}(n^2 - nk)$ entries), are called *low-density parity-check codes* (LDPC codes). The codes which we constructed out of expanders are called *expander codes*.

We will first prove claims about the parameters of expander code. Since the parity check is $n \times m$ matrix, the code has domension $k = n - m$. We prove the following claim about the distance of the code.

LEMMA 16.1.2. *The expander code constructed above has minimum distance at least αn.*

Proof. The code is linear by definition. It suffices to show that every codeword has at least αn number of 1s in them. Suppose a codword c has less than αn number 1s. We can associate a subset of vertices in U corresponding to the indices where the 1 appears in c.

$$S = \{i \in [n] \mid c_i = 1\} \subseteq U$$

By definition $|S| < \alpha n$. Hence expansion property applies. But then, by lemma 22.1.1, there must be at least $D/2$ unique neighbors. This would imply that $Hc \neq 0$ and hence c is not a code word. ∎

16.2 Unique Decoding Expander Codes

Now we will talk about decoding this code. For a node $v \in V$, if y was a codeword, we would have :

$$P_v = \bigoplus_{j \in N(v)} y_j = 0$$

The algorithm is surprisingly simple. We describe the algorithm first: as for notation, for a node $u \in U$:

$$\Gamma(u) = \{v \in N(u) \mid P_v = 1\}$$

We call u to be *likely-to-be-corrupted* if $|\Gamma(u)| > \frac{D}{2}$. In this case, intuitively, it is likely that u was a corrupted bit and hence it seems reasonable to flip u as a part of the error correction. We do this systematically in the following algorithm:

Algorithm 16.23 : Unique Decoding Algorithm for Combinatorial Expander Codes

Input: $y \in \{0,1\}^n$ such that $d(y, C) < \frac{\alpha n}{2}$
Output: The unique code c such that $d(c, y) < \frac{\alpha n}{2}$

1: While ($\exists u \in U$, such that $|\Gamma(u)| > \frac{D}{2}$)
2: Flip the value of y_u.

The termination of the algorithm is clear since each time we execute step 2, the number of unsatisfied vertices on the right would have strictly decreased.

LEMMA 16.2.1. *The Algortihm 22.23 terminates and recovers the codeword from received word y with guarantee $d(y, C) < \frac{\alpha n}{2}$.*

Proof. Let S be the set of flipped bits in the received word y. For a vertex $v \in V$ to be unsatisfied it must me a neighbot to at least one of the bits in S. Thus we have at most $\frac{\alpha n D}{2}$ many unsatisfied vertices in V. Since each step of the algorithm strictly decreases the number of unsatisfied vertices in V, the algorithm must terminate in $O(n)$ steps.

We need to show that we will end up with a codeword in the end - that is the algorithm never gets stuck - if y is not a codeword then there is always a bit that will need to be flipped as per the rules of the algorithm. For such a word y, we know that $|S| \geq 1$. There are $D|S|$ many neighbors for S which we call T. More than $\frac{D}{2}|S|$ of the elements in T are unique neighbors and hence they will all be unsatisfied. By averaging, there must be a vertex $u \in S$ such that at least $\frac{D}{2}$ of these unsatisified vertices are adjacent to u. Given that the total number of neighbors for u is D, this gives that u has more than half of its neighbors unsatisfied and hence is the vertex which will get flipped by the algorithm. □

Curiosity 16.2.2 (Speilman Codes). The running time of the decoder was $O(n^2)$ time (we showed only $O(n)$ rounds bound in class). With the right kind of data structure it can be improved to work in $O(n)$ time. It turns out that encoding is hard to do efficiently, since only the paritycheck matrix is defined based on the graph. Spielman came up with the *superconcentrator* based codes, which have both linear time encoding and linear time decoding.

16.3 Tanner Codes

We introduce the idea of Tanner Codes now. Tanner in the 1980s discovered codes based on bipartite graphs. If C is an $[D, k, \delta D]_2$ code, and $G(U, V, E)$ is a bipartite graph with both parts having n vertices each, which is D-right-regular. The Tanner Code $T(C)$ is the set of codewords defined by :

$$T_G(C) = \left\{ c \in \mathbb{F}_2^n \mid \forall v \in V, c|_{N(v)} \in C \right\}$$

where $c|_{N(v)}$ denotes the subsequence formed by the bits indexed by the neighbors of v in the graph G. Tanner codes are linear codes because the set of codewords are defined by subwords being in linear space C. They are also generalizations of expander codes since in expander code C was chosen to be the $[D, D-1, 2]_2$ parity check code.

To determine the dimension of the code: the constraint that each of the $c|_{N(v)}$ must be in C (which is a space of dimension k inside a space of dimension D) is equivalent to writing $D-k$ constraints. Thus, we have introduced a total of $2n(D-k)$ constraints and hence the domension is $n-n(D-k)$.

In expander codes which are a special case of Tanner codes, we used parity check codes for C. Since parity check codes have distance 2, we required that all sets of size at most αn expand by a factor of more than $\beta > \frac{3}{4}$ in order to argue that the code had distance at least αn. However, if we use a local code C of distance d, then we only require an expansion factor exceeding $\frac{3}{2d}$ to ensure the same code distance. Hence a good choice of C allows us to construct explicit Tanner codes using graphs with weaker expansion properties which are therefore easier to construct.

16.4 Tanner Codes from Spectral Expanders

Recall the definition of a spectral expander:

DEFINITION 16.4.1. *A graph $G=(V,E)$ is said to be a (N,D,λ)-expander if G is a d-regular graph on n vertices and $|\lambda_1| \geq |\lambda_2| \geq \ldots \geq |\lambda_n|$ are the eigenvalues of the adjacency matrix of G.*

We need bipartite graphs for the construction. Hence we consider the bipartite version of the graph, which is sometimes called the *double cover* of the graph. It has two copies of each node of G, one in the left partition and the other in the right partition, and there are two copies of each edge (u,v) of G - one from the vertex u on the left to vertex v on the right, and the other vice versa. Note that the bipartite adjacency matrix of the double cover is exactly the adjacency matrix of the original undirected graph. From now on, we will denote by G this bipartite cover.

We use the expander mixing lemma stated for the double cover graphs.

LEMMA 16.4.2 (**Expander Mixing Lemma**). *Let $G(U,V,E)$ be a D-regular bipartite graph with n vertices and let $\lambda_2 < 1$ be the second largest eigen value[36] of the normalized bipartite adjacency matrix of G. Then, for any $S \subseteq U$, $T \subseteq V$:*

$$\left| |E(S,T)| - \frac{D|S||T|}{n} \right| \leq D\lambda_2 \left(\sqrt{|S||T|} \right)$$

We start with a D-regular bipartite graph on N vertices on either side, where the graph is symmetric. Let the second largest eigen value of the bipartite adjacency matrix be λ. Consider C by an $[D,k,\delta n]$ code over $\mathbb{F}$ with rate $R=\frac{k}{D}$. We obtain a new code C' which is an $[N',k',\delta' n]$ code from it over $\mathbb{F}$ were $N'=ND$.

We need to define which all vectors in $\mathbb{F}^{ND}$ are in the code space C'. Consider a vector $c \in \mathbb{F}^{ND}$. We can associate each component of the vector c with the pair (a,i) where $a \in U \cup V$ and $i \in [D]$. That is, view c as the sequence:

$$c = (c_{u,i} : u \in U \text{ and } i \in [D]) = \left(c_{v,j} : v \in V \text{ and } j \in [D]\right)$$

under appropriate re-ordering of the indices. The given vector c is a codeword of C' if and only if

[36] Note that the double cover graphs are symmetric : $(u,v) \in E \Rightarrow (v,u) \in E$. Hence all eigen values are real.

for every $a \in U \cup V$, the word $c_a = (c_{a,i} : i \in [D])$ is a codeword in C.

Parameters of the new code: The blocklength of the code is $N' = ND$. To determine the dimension of the code: the constraint that each of the w_a must be in C (which is a space of dimension k inside a space of dimension D) is equivalent to writing $D - k$ constraints. Thus, we have introduced a total of $2N(D-k)$ constraints and hence the domension is $ND - 2N(D-k)$. Hence the rate of the code is:

$$\frac{ND - 2N(D-k)}{ND} = \frac{2k}{D} - 1 = 2R - 1$$

Thus the rate of the code improved by this operation. We now compute the minimum distance.

LEMMA 16.4.3. *The relative distance of the code C' is $\delta(\delta - \lambda)$ where λ is the second largest eigen value of the graph G.*

Proof. Since the code is linear, it suffices to lower bound the weight of non-zero codewords. Consider any non-zero codeword $w \in C'$, and consider the positions of non-zero symbols in it. Define this index set as follows:

$$E_U = \left\{ (u,i) : \begin{array}{c} u \in U, i \in [D] \\ w_{u,i} \neq 0 \end{array} \right\} \qquad E_V = \left\{ (v,i) : \begin{array}{c} v \in V, i \in [D] \\ w_{v,i} \neq 0 \end{array} \right\}$$

Note that E_U and E_V can be interpreted as a set of edges in the graph G and prove a lower bound on the distance, it suffices to prove a lower bound on the size of $E = E_U \cup E_V$ as a multiset.

Let S and T be the set of vertices that are incident to E_U and E_V respectively. Since w_a for each $a \in S \cup T$ are non-zero codewords in C, each $u \in S$ (resp. $v \in T$) must have at least δD edges in E_U (respectively in E_V) incident on it. If we define $E = E_U \cup E_V$, as a multi-set, $|E| \geq \delta D|S|$ and $|E| \geq \delta D|T|$ and hence $|E| \geq \delta D\sqrt{|S||T|}$.

Applying expander mixing lemma for this pair of subsets of vertices S and T gives,

$$\delta D\sqrt{|S||T|} \leq |E| \leq \frac{D|S||T|}{N} + \lambda D\sqrt{|S||T|}$$

and hence,

$$\sqrt{|S||T|} \geq (\delta - \lambda)N$$

This gives,

$$|E| \geq \delta(\delta - \lambda)DN$$

Hence the relative distance of the code is at least $\delta(\delta - \lambda)$. ∎

REMARK 16.4.4. Note that when G is the complete bipartite graph, the code C' that we obtain is simply the product of C with itself (that is each symbol in the codeword is replaced by an entire codeword of the same code) and thus has relative distance exactly δ^2 and blocklength D^2. By using expander, we can get roughly the same distance, but at the same almost double the rate, as opposed to squaring in the case of product construction (note that $R^2 < R$).

16.5 Unique Decoding Spectral Expander Codes

Suppose the code C was able to correct up to ϵ fraction of errors. The algorithm for decoding of C' is super simple to describe. In each round, process at all the vertices on the left of the bipartite graph and decode the edges there to codewords in C and replace the symbols. Repeat the same operation from the perspective of the vertices on the right. We describe this formally as a pseudocode in Algorithm 22.24.

Algorithm 16.24 : Unique Decoding Algorithm for Spectral Expander Codes

Input: $y \in \mathbb{F}^{ND}$ such that $d(y, C) < \eta ND$
Output: The unique code c such that $d(c, y) < \eta ND$.

1: **repeat**
2: **Left-side Decoding Step:**
3: **for** $u \in U$ **do**
4: Correct $y_u \in \mathbb{F}^D$ by using the decoding algorithm for C.
5: Replace the symbols on the edges accordingly.
6: **end for**
7: **Right-side Decoding Step:**
8: **for** $v \in V$ **do**
9: Correct $y_v \in \mathbb{F}^D$ by using the decoding algorithm for C.
10: Replace the symbols on the edges accordingly.
11: **end for**
12: **until** No changes happen for an entire iteration

LEMMA 16.5.1. *The above decoding algorithm can recover from $\frac{9}{10}\epsilon(\epsilon - \lambda)$ fraction of errors, in $O(N \log N)$ time.*

Proof. Suppose a received word $y \in \mathbb{F}^{|E|}$ is given to us. As per our interpretation, we are associating each symbol of y to edges of two symbols of y to each edge of the graph G correspoding to either endpoints. We use the terminology that a vertex is *erroneous* if the value associated with at least one edge incident to the node is incorrect in comparison to the correct codeword.

Let $S \subseteq U$ denote the set of vertices in U that are erroneous before the first left-decoding step (steps 2-6), and let $T \subseteq V$ be the set of vertices in V that are erroneous after the next right-decoding step (steps 7-11). Let $E' \subseteq E$ be the set of *edges* that has error at the step in between the two decoding steps (step 6). Note that $E(S, T) \leq E'$ since the edges that has errors has be definition an edge to S.

Since we are using the decoding algorithm for C, we know that if the decoding algorithm for C run in step 4 could not correct the word, then it must be because the number of errors is more than ϵD. Thus, the number of errors in the original word must be at least $|S|\epsilon D$.

$$|S|\epsilon D \leq \frac{9}{10}\epsilon(\epsilon - \lambda)ND \quad \text{and this gives an upper bound } |S| \leq \frac{9(\epsilon - \lambda)N}{10}$$

Again, similarly, we know that if the decoding algorithm for C run in step 9 could not correct the

word, then it must be because the number of errors is more than ϵD. This gives,

$$\begin{aligned}
|T|\epsilon D \leq |E'| \leq |E(S,T)| &\leq \frac{D|S||T|}{N} + \lambda\sqrt{|S||T|} \\
&\leq \frac{D|S||T|}{N} + \lambda\frac{|S|+|T|}{2} \\
&\leq \frac{9D|T|}{10}(\epsilon-\lambda) + \lambda\frac{|S|+|T|}{2} \\
|T|\left(0.1\epsilon D + 0.9D\lambda - 0.5D\right) &\leq 0.5\lambda|S| \\
|T| &\leq \frac{\lambda|S|}{0.2\epsilon D + 1.8D\lambda - \lambda} \leq \alpha|S|
\end{aligned}$$

where $\alpha < 1$ is a constant, if $\lambda < \epsilon D$. That is, we have an initial code that can correct at least $\frac{\lambda}{D}$ fraction of errors. Thus after each iteration, the number of erroneous vertices goes down by a constant factor α. Since it is at most a constant fraction of N initially, in $O(\log N)$ steps we would converge to 0 errors. In each step, we need $O(N)$ steps and this makes the algorithm $O(N \log N)$ time. □

Part II

Exercise & Problem Sets

Chapter 17

Exercises

17.1 Exercises

Exercise 17.1.1. Let $G \in G(n,p)$. For all $S \subseteq V$, let A_S be the event that S forms an independent set in G. Show that if S and T are two distinct subsets of k vertices then A_S and A_T are independent if and only if $|S \cap T| \leq 1$.

Exercise 17.1.2. Prove that an event $\mathcal{E}$ is independent of a set of events $\{\mathcal{E}_j \mid j \in J\}$ if and only if for all $J_1, J_2 \subseteq J$ such that $J_1 \cap J_2 = \phi$

$$\Pr[\mathcal{E} \cap (\cap_{j\in J_1} B_j) \cap (\cap_{j\in J_2} \overline{B_j})] = \Pr(\mathcal{E}) \Pr[(\cap_{j\in J_1} B_j) \cap (\cap_{j\in J_2} \overline{B_j})]$$

Exercise 17.1.3. Suppose $k > 2$ and let H be a k-uniform hypergraph with 4^{k-1} edges. Show that there is a 4-colouring of $V(H)$ such that no edge is monochromatic.

Exercise 17.1.4. For a set of Boolean variables $x_1, x_2, \ldots x_n$, a k-CNF formula ϕ has the form $\phi = C_1 \wedge \ldots \wedge C_m$. Each clause C_j is an or of some set of k literals, where each literal is either x_i or x_i for some $i \in [n]$. Clauses C_j and C_k are said to intersect if $\exists x_i$ such that both clauses contain either x_i or $\neg x_i$. A satisfying assignment is a setting of the x_is that makes ϕ evaluate to true. Deciding the existence of a satisfying assignment for a Boolean CNF formula is another well-studied (and NP-complete) problem. This problem asks you to show that a satisfying assignment exists in certain cases. If each clause intersects at most $\frac{2^k}{e} - 1$ other clauses, then show that ϕ is satisfiable. (Hint: Use LLL).

Exercise 17.1.5. Let X be a random variable and $\mathbb{E}[X] = \mu$ be the expectation. Variance captures the expected square deviation from the mean μ. That is $\mathsf{Var}[X]$ is defined as $\mathbb{E}[(X-\mu)^2]$. It can be shown to be equal to $\mathbb{E}[X^2] - (\mathbb{E}[X])^2$ and that $\mathsf{Var}[aX] = a^2\mathsf{Var}[X]$. And more importantly, when the random variables $X_1, X_2, \ldots X_n$ are pairwise independent, then show that:

$$\mathsf{Var}[X_1 + X_2 + \ldots + X_n] = \mathsf{Var}[X_1] + \mathsf{Var}[X_2] + \ldots + \mathsf{Var}[X_n]$$

(Hint: Use the technique used in the proof of Lemma 5.2.1)

Provide an example that shows that the variance of the sum of two random variables is not necessarily equal to the sum of their variances, when the random variables are not independent. Indeed, one dramatic example when $n = 2$ is to take $X_2 = -X_1$. Find less dramatic ones !.

Exercise 17.1.6. Recall the discussion on Ramsey graph's where we applied probablistic method to show bounds on Ramsey number $R(k,k)$. Let us define a graph to be a *k-Ramsey graph* if it has no clique or independent set of size at least k. We had shown by the probabilistic method that there exists a graph on n vertices that $O(\log n)$-Ramsey. An interesting open problem is to give an *explicit* construction of a $O(\log n)$-Ramsey graph. The best known construction runs in time $O(2^{2^{\log^{\epsilon} n}})$ for some small $\epsilon = 0.99$. In this problem you are asked give a construction for $O(\log n)$ Ramsey graph that runs in time $O(n^{\log^2 n})$. [Hint : Check if full independence of random choices is necessary in our proof by probabilistic method. Show that the proof actually gives a set of $O(n^{\log^2 n})$ graphs, one of which is guaranteed to be $O(\log n)$-Ramsey]

Exercise 17.1.7. Let $n, d \in \mathbb{N}$, $\alpha, \beta > 0$. Every (n,d,α,β)-boundary-expander is also a $(n,d,\alpha,\frac{\beta}{d})$-edge-expander. Conversely, every (n,d,α,β)-edge-expander is also a (n,d,α,β)-boundary-expander.

Exercise 17.1.8. Prove Claim 6.5.2 for a general α.

Exercise 17.1.9. If A is the normalized adjacency matrix of G and $x \in \mathsf{R}^n \setminus \{0\}$:

$$\sum_{i,j} A_{ij}(x_i - x_j)^2 = 2x^T x - 2x^T A x$$

Exercise 17.1.10. If A has eigen values $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$, and B has eigen values $\lambda'_1 \geq \lambda'_2 \geq \ldots \geq \lambda'_m$, then $A \otimes B$ has eigen values as :

$$\{\lambda_i \lambda'_j \mid 1 \leq i \leq n, 1 \leq j \leq m\}$$

Exercise 17.1.11. For the paramter r used above, Hadamard code and Simplex code both have distance 2^{r-1}. In fact something stronger is true, every non-zero codeword has weight exactly 2^{r-1}.

Exercise 17.1.12. Let $G = (V,E)$ be any undirected graph (assume no loops or multiple edges). A cut in the graph is the subset of all edges that connect a vertex in S to vertex in $V \setminus S$, for some subset $S \subseteq V$. Let $\mathsf{Cuts}(G) \subseteq \{0,1\}^E$ consist of the characteristic vectors of all cuts of G.

(a) Prove that $\mathsf{Cuts}(G)$ is an $[|E|, |V|-1]_2$ binary linear code. What parameter of G equals the distance of Cuts(G)?

(b) Describe the dual code $\mathsf{Cuts}(G)^{\perp}$ of $\mathsf{Cuts}(G)$. What is its dimension? What parameter of G equals the distance of $\mathsf{Cuts}(G)^{\perp}$?

Exercise 17.1.13. Assume that there exists an $[n,k,d]_q$-code C over $\mathbb{F}_q$. Then:

(a) (**Extension**) There exists an $[n+r,k,d]_q$ code for every $r \geq 1$.

(b) (**Puncturing**) There exists an $[n-r,k,d-r]_q$ code for every $r \in [d-1]$.

(c) There exists an $[n,k,d-r]_q$-code over $\mathbb{F}_q$ for every $1 \leq r \leq d-1$.

(d) (**Subcode**) There exists an $[n,k-r,d]_q$ code for every $r \in [k-1]$.

(e) There exists a $[n-r, k-r, d]_q$ code for every $r \in [k-1]$.

(f) (**Direct Sum**) Let C_1, C_2 be linear codes such that for $i \in [2]$, C_i is an $[n_i, k_i, d_i]_q$. Define the direct sum :

$$C_1 \oplus C_2 = \{(c_1, c_2) \mid c_1 \in C_1, c_2 \in C_2\}$$

is an $[n_1 + n_2, k_1 + k_2, \min\{d_1, d_2\}]_q$-code. Write down the generator matrix for the direct sum code.

(g) Let C_1, C_2 be linear codes such that for $i \in [2]$, C_i is an $[n, k_i, d_i]_q$. Define the direct sum :

$$C = \{(c_1, c_1 + c_2) \mid c_1 \in C_1, c_2 \in C_2\}$$

is an $[2n, k_1 + k_2, \min\{2d_1, d_2\}]_q$-code. (Hint: Analyse in two cases : $c_2 = 0^n$, $c_2 \neq 0^n$.)

(h) Let $\overline{x}$ denote the bitwise complement of a vector x, and let C be a $[n, k, d]_2$-code. Then the code:

$$C' = \{(c, c), (c, \overline{c}) \mid c \in C\}$$

is a $[2n, k+1, \min\{2d, n\}]_2$-code. (Hint : Repetition code) This provides us a way of rovides us a way to double all parameters (n, k and d).

Exercise 17.1.14. For a bivariate polynomial of degree t, the number of monomials given by

$$|\{(i, j) | 0 \leq i, j \leq t, i + j \leq t\}|$$

is $\binom{t+2}{2}$. In general show that for a v variate polynomial of degree t, number of monomials is $\binom{t+v}{v}$.

Exercise 17.1.15. Argue that the Reed-Muller code is linear.

Exercise 17.1.16. Let C be an $[n, k, d]_q$ code. Let $y = (y_1, \ldots, y_n) \in (\mathbb{F}_q \cup \{?\})^n$ be a received word (where ? denotes an erasure) such that $y_i =?$ for at most $d-1$ values of i. Present an $O(n^3)$ time algorithm that outputs a codeword $c = (c_1, \ldots, c_n) \in C$ that agrees with y in all unerased positions (i.e., $c_i = y_i$ if $y_i \neq ?$) or states that no such c exists.(Recall that if such a c exists then it is unique.)

Exercise 17.1.17 (Protocol using Hadamard Codes). Come up with a protocol for $\text{EQ}(x, y)$ function with cost $O(\log n)$ bit of communication, using Hadamard Codes.

17.2 Curiosity Drive

Here we list down all the "out of curious" questions that we discussed (sometimes even not discussed) in the class (and hence in this document).

Curiosity 17.2.1 (Property-B Conjecture). A hypergraph H has **Property B** (or 2-colorable) if there is a red-blue vertex-coloring with no monochromatic edge. A hypergraph with property B is also called bipartite, by analogy to the bipartite graphs. Erdos (1963) asked: What is the minimum number of edges $m_2(k)$ of a k-uniform hypergraph not having property B? Indeed, the above discussion implies that $m_2(k) \geq 2^{k-1}$. Erdos proved an upper bound of $m_2(k) \leq O(k^2 2^k)$. The best known bounds are :

$$\Omega\left(\sqrt{\frac{k}{\ln k}}2^k\right) \leq m_2(k) \leq O\left(k^2 2^k\right)$$

The upper bound is due to Erdos (1964) and the lower bound went through a series of improvements to reach the above bound Radhakrishnan and Srinivasan (2000).

$$\left\{\begin{array}{c} m_2(k) \geq \left(\frac{1}{2}\right)2^k \\ \text{[Erdos, 1963]}\end{array}\right\} \rightarrow \left\{\begin{array}{c} m_2(k) \geq \left(\frac{k}{k+4}\right)2^k \\ \text{[Schmidt, 1964]}\end{array}\right\} \rightarrow \left\{\begin{array}{c} m_2(k) \geq \left(\sqrt[3]{k}\right)2^k \\ \text{[Beck, 1978]}\end{array}\right\} \rightarrow \left\{\begin{array}{c} m_2(k) \geq \left(\sqrt{\frac{k}{\ln k}}\right)2^k \\ \text{[RS, 2000)]}\end{array}\right\}$$

It is believed that $k2^k$ is the right asymptotic bound for $m_2(k)$. In fact, this is a conjecture due to Erdos and Lovasz: $m_2(k) \in \Theta(k2^k)$.

Curiosity 17.2.2. The best approximation ration for maximum cut problem is not $\frac{1}{2}$. It is 0.878, and it comes from the semidefinite programming (SDP) relaxation for maxcut problem LP. Define $n+m$ variables x_u for each $u \in V$ and $e_{uv} \in E$. These variables are supposed to represent the information $e_{uv} = 1$ if and only if (u,v) is in the cut, and $x_u = 1$ if and only if $u \in S$. The LP objective function is to maximise $\sum_{(u,v)\in E} e_{uv}$ subject to the constraints (1) $\forall u,v \in E$, $e_{uv} \leq x_u + x_v$ and $e_{uv} \leq 2 - (x_u + x_v)$, (2) all of them are Boolean variables. The idea due to [Goemans and Williamson, 1995] is to relax this condition (2) by using vector valued variables (rather than Boolean). Not only that this relaxation can be solved efficiently, but [Goemans and Williamson, 1995] gave a method of rounding the variables to Boolean values and proving that the approximation ratio is atleast 0.878 . . ..

Complementing this, an optimal inapproximability result was proven for MAXCUT by [Khot et al., 2007], based on a conjectured hardness for the approximation problem known as the label cover problem (this is also called the *unique games conjecture* (UGC) - see [Khot, 2010]).

Curiosity 17.2.3.

7: Jayalal says: Todo - Write about Luby's algorithm

Curiosity 17.2.4 (Gershgorin's Circle Theorem). This is a digression. We used the following statement (where we left the proof as an exercise). *The largest eigen value of a doubly stochastic matrix is* 1 For graphs without self-loops, this is a special case of a more general theorem called *Gershgorin circle theorem* may be used to bound the spectrum of a square matrix. It was first published by the Gershgorin in 1931. The statement is as follows:

Let A be an $n \times n$ matrix with entries from $\mathbb{C}$, with entries a_{ij}. For $i \in [n]$ let $R_i = \sum_{j \neq i} |a_{ij}|$ be the sum of the absolute values of the non-diagonal entries in the i-th row. Let $D(a_{ii}, R_i) \subseteq \mathbb{C}$ be a

closed disc centered at a_{ii} with radius R_i. Such a disc is called the *Gershgorin disc*.

Gershgorin's Circle Theorem : *Every eigenvalue of A lies within at least one of the Gershgorin discs.*

Indeed, in our case, for every i, $R_i = 1$, $a_{ii} = 0$. This immediately implies the statement we wanted to conclude. For a diagonal matrix, the Gershgorin discs coincide with the spectrum.For a diagonal matrix, the Gershgorin discs coincide with the spectrum. Conversely, if the Gershgorin discs coincide with the spectrum, the matrix is diagonal.

Curiosity 17.2.5 (Converse of Expander Mixing Lemma). Expander Mixing Lemma (Lemma 6.8.1) says that the edges acorss S and T for any two subsets of matrices behaves like that of the random graphs. In fact, even the converse is also true. For any two subsets of vertices if the density of the subsets of matrices behaves like that of random graphs, then it must necessarily have a good spectral gap.

In fact, the converse of Expander Mixing Lemma (Lemma 6.8.1) is also known [Bilu and Linial, 2006]. The statement is as follows: Let G be a d-regular graph and suppose that

$$\left| |E(S,T)| - \frac{d|S||T|}{n} \right| \leq d\delta \left(\sqrt{|S||T|} \right)$$

then $\lambda_2(G)$ is $O\left(\delta + \delta \log\left(d/\delta\right)\right)$.

Curiosity 17.2.6 (Ramanujan Graphs). Ramanujan graph, is a regular graph whose spectral gap is almost as large as possible. Such graphs are excellent spectral expanders. The complete graph K_{d+1} has spectrum $d, -1, -1 \ldots -1$ and thus $\lambda_2(K_{d+1}) = d$ and hence is a Ramanujan graph for every $d > 1$. The complete bipartite graph $K_{d,d}$ has spectrum $d, 0, 0, \ldots 0, -d$ and hence is a bipartite Ramanujan graph for every d. [Lubotzky et al., 1988] showed[1] how to construct an infinite family of $(p+1)$-regular Ramanujan graphs, whenever p is a prime number and $p \equiv 1 \mod 4$. This was extended for any prime power later.See [Murty, 2003] for a survey. It is still an open problem whether there are infinitely many d-regular (non-bipartite) Ramanujan graphs for any $d \geq 3$.

Curiosity 17.2.7. The above is a trivial bias amplification method and the analysis depends on independence of the sampling. One may ask the question, can we do more randomness efficient amplification of the bias?

Given our background with expanders, it is natural to consider a random walk on expanders. A simple case is consider the following length 2 walk lemma which is attributed to Rozenman and Wigderson (unpublished).

LEMMA 17.2.8. *Let D be an ϵ-biased distribution over $\{0,1\}^m$ and let G be an $(2^m, d, \lambda)$ spectral expander whose vertices are labeled by samples of D so that the number of vertices labeled $w \in \{0,1\}^m$ is proportional to $D(w)$. Let D' be the following distribution: Uniformly choose a random vertex y_1 and choose a random neighbor of y_2 to sample a random edge (y_1, y_2) of G and output $y = y_1 + y_2$. Then D' is $(\epsilon^2 + \lambda)$-biased and with support $O(d.\mathsf{supp}(D))$.*

Notice that ϵ^2 is smaller than ϵ and that amplifies the bias, and we lose it out a bit by additive λ. Thus if we choose a better spectral expander, we achieve amplification without much loss. One can

[1] The proof uses what is called Ramanujan conjecture, which led to the name of Ramanujan graphs.

also repeat this again and again, and achieve a good bias amplification for resulting distribution D_i which will be ϵ_i-biased and support size s_i such that $\epsilon_{i+1} = \epsilon_i^2 + \lambda_i$ and $s_{i+1} = O(d.s_i)$. Choosing $\lambda_i = \epsilon_i^2$ and $d_i = 1/\lambda_i^4$, this gives:

$$\epsilon_{i+1} = 2\epsilon_i^2 \text{ and } s_{i+1} = s_i/\epsilon_i^4$$

Instead of repeating this on different graphs, one can imagine [Ta-Shma, 2017] taking a walk for t steps: that is, choose y_1 from D and then take a walk on the s-wide replacement product reduces the bias almost optimally. If the labels obtained in the walk be $y_1, y_2, y_3, \ldots y_t$ then output $y = y_1 + y_2 + \ldots + y_t$. Let D^t be the resulting distribution whose bias we need to estimate. [Ta-Shma, 2017] analyses this as follows: let $w \in \{0,1\}^m$ and $w \neq 0$. We need to show that:

$$\Pr_{y \sim D^t}[\langle y, w\rangle = 1] = \Pr_{y \sim D^t}\left[\sum_{i=1}^{t} \langle y_i, w\rangle = 1\right] \in \left[\frac{1}{2} - \frac{\delta}{2'}, \frac{1}{2} + \frac{\delta}{2}\right]$$

Thus, if we define a bad set B as follows:

$$B = \left\{ u \in \{0,1\}^m \mid \sum_{i:w_i=1} u_i = 1 \right\}$$

We need to estimate the probability that $\sum_{i=1}^{t} \langle y_i, w\rangle = 1$. This is same as the probability that an odd number of y_is sampled falls into the bad set B. Compare this with the expander walk based error reduction method, where we wanted to analyse the probability that the majority of the samples fall into a bad set. It is surprising that a similar analysis works for the "parity" of the samples to be in B also. We refer the reader to [Ta-Shma, 2017] for modeling this interesting fact algebraically and a proof. See Section 3 of [Ta-Shma, 2017].

Curiosity 17.2.9 (Concatenation of Codes). A natural question that may arise is why could we not just encode the q-ary alphabet symbols using binary alphabet "inside" the code and then use it as a binary code. One immediate difficulty with this is that one bit change in the encoding can change the whole symbol - hence 1 bit error can potentially have the effect of $O(\log q)$ errors. This will make the parameters quite weak. Then, how about using another code inside the binary encoding so that we can correct those errors too. While it may look complicated, it is possible do this systematically and construct a code with a small alphabet from a code with a large alphabet. Let q by a prime power, and let C be an $[N, K, D]_{q^k}$ code and let C' be an $[n, k, d]_q$ code . Then there exists a linear code C'' over $\mathbb{F}_q$ with parameters $[nN, kK, d']_q$ such that $d' \geq dD$. C is called the *external* code, and C' is called the *internal* code.

This is particularly interesting to apply for concatenation two codes with good distances. Consider the Reed-Solomon codes which requires large alphabet size, and the Hadamard code. The external code is the Reed-Solomon code $[n, \frac{n}{2}, \frac{n}{2} + 1]$ over $\mathbb{F}_q$ with $q = n = 2^r + 1$ for some r. We then concatenate this code with the Hadamard code with parameters $[2^r, r + 1, 2^r - 1]$. In terms of n the inner code is $[\frac{n}{2}, \log n, \frac{n}{4}]$. This concatenation is legal because the block length of Reed-Solomn code and the message length of Reed-Muller code exactly matches. By the above theorem (in the previous paragraph), we have that the concatenated $[\frac{n^2}{2}, \frac{n}{2}\log n, \frac{n^2}{8}]$. But unfortunately this codes does not have constant rate.

The problem with concatenating Reed-Solomon codes with Hadamard code is that the internal code does not have constant rate. We would therefore like to concatenate the Reed-Solomon code with a internal binary code with constant rate and distance. One can choose codes which achieves the Gilbert-Varshamov bound. But then, we do not know if such a code that is explicitly constructible. Even more interestingly, **[?]** asked *do we require inner code to be explict and efficiently encodable/decodable?* The answer is no, because the codewords are symbols of the external code and the internal codes message lengths are logarithmic in the blocklength. Thus, if the internal code can be constructed in time that is exponential in the message length of the code, then this will still be polynomial in the length of the concatenated code and still good enough for us. The asymptotically good codes (costant rate and relative distance) constructed out of this idea are called Forney Codes **[?]**.

Curiosity 17.2.10 (Reed-Solomon Code for CDROMs). (This is a description from a lecture notes by Yehuda Lindell) CIRC stands for *Cross Interleaved Reed-Solomon code*; it was developed in 1979 by Phillips and Sony (the CD-ROM standard includes the error correcting code). The code provides a very high level of error correction and is the reason that even some scratched CDs typically keep working. This is a demonstration of the fact that the knowledge obtained in this part of the course is enough to understand real codes that are used.

Every disk contains a spiral track of length approximately 5 km. The track contains pits and lands (for representing bits). The width of the track is $0.6\mu m$ and the depth of a pit is $0.12\mu m$. The laser is shone onto the track and is reflected with a different strength if it is focused on a land or a pit, enabling the bit to be read. The errors are usually burst errors, meaning that they come in a continuous region.

Every audio sample is 16 bits long; in order to achieve stereo, two samples are taken for each time point. There are 44,100 samples per second and every sample pair is 4 bytes long. Therefore, every second of audio is made up of 176,400 bytes. Each sample in a pair is a vector in $\mathbb{F}_2^{16}$; each such vector is divided into two, and we view each half as an element of F_2^8. Two Reed-Solomon codes, denoted C_1 and C_2 are used in an interleaved fashion (this results in distributing large burst errors into small local errors).

Curiosity 17.2.11 (Guruswami-Sudan Algorithm). To write the ideas required to improve the above agreement constraint to $t > \sqrt{nk}$ which matches (in relative to n) with the Johnson radius of $1 - \sqrt{R}$ for Reed-Solomon codes.

8: Jayalal says: Todo - Yet to write about Guruswami-Sudan Algorithm

Curiosity 17.2.12 (Reselient Algorithms for Computing Polynomials). The framework that we discussed is applicable to the problem of correcting programs that compute multivariate polynomials over large finite fields and not just computing the permanent polynomial. This gives an efficient procedure to transform any program that computes a multivariate polynomial $P(x)$ correctly on a $1/2 + \epsilon$ fraction of its inputs ($\epsilon > 0$) into a randomized program that computes $P(x)$ correctly on every input with constant probability. In fact, this is the result proved by **[?]** and the argument presented in this lecture is a corollary of their proof.

Curiosity 17.2.13 (Speilman Codes). The running time of the decoder was $O(n^2)$ time (we showed only $O(n)$ rounds bound in class). With the right kind of data structure it can be improved to work

in $O(n)$ time. It turns out that encoding is hard to do efficiently, since only the paritycheck matrix is defined based on the graph. Spielman came up with the *superconcentrator* based codes, which have both linear time encoding and linear time decoding.

Chapter 18

Problem Sets

18.1 Problem Set #1

(1) (See Exercise 2.1.6) A random k-colouring fo a graph G is an element of the probability space (Ω, Pr) where Ω is the set of all k-colourings (i.e. partition of V into k sets $(V_1, V_2, \ldots, V_k)$, all this colourings being equally likely (so happening with probability $\frac{1}{k^n}$. For every edge e of G, let A_e be the event that the two endvertices of e receive the same colour. Show that:

(a) for any two edges e and f of G, the events A_e and A_f are independent.

(b) if e, f and g are three edges of a triangle of G, the events A_e, A_f and A_g are dependents.

(2) (See Exercise 2.1.7) A graph $G = (V, E)$ is created at random by selecting each edge with probability p. What is the expected number of spanning trees in the randomly sampled graph? (*Hint : Use Cayley's Theorem that the number of distinct spanning trees on n vertices is n^{n-2}. Order them, and define an indicator random variable.*)

(3) (See Exercise 2.2.1) In the derandomization of MAXCUT algorithm that we described, we derived an expression for $V(r_1, r_2, \ldots r_i)$ for any i and $r_1, r_2, \ldots r_i \in \{0, 1\}$. We used this to determine, the value of r_{i+1} by computing $V(r_1, r_2, \ldots r_i, 0)$ and $V(r_1, r_2, \ldots r_i, 1)$ and then choosing the value of r_{i+1} to be the one which produces the largest among the two. Prove that the choice of r_{i+1} will be 1 if vertex $i+1$ has more neighbors in T_i than in S_i and vice versa. Hence, write down the derandomized 0.5-approximation deterministic polynomial time algorithm for MAXCUT as a simple greedy algorithm in terms of the above rule.

(4) (See Exercise 2.2.2) Let $x_1, x_2, \ldots x_n \in \{0, 1\}$ be Boolean variables and let f be a Boolean formula in CNF form. That is, $f = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ where each C_i (called a *clause*) is a disjunction of literals in the set $\{x_1, \overline{x_1}, x_2, \overline{x_2}, \ldots, x_n, \overline{x_n}\}$. We want to find an assignment of the Boolean variables that satisfies as many clauses in the formula as possible.

(a) Write down a randomized algorithm that outputs an assignment with the guarantee that the expected number of clauses satisfied is at least $\frac{m}{2}$.

(b) Derandomize this algorithm using the method of conditional probabilities discussed in class to get a deterministic algorithm that satisfies at least $\frac{m}{2}$ number of clauses.

(c) Suppose k is the minimum number of literals in any clause, how will you modify the parameters in part(a) and (b)

(5) (See Exercise 3.1.3) A decision problem L is in a class called BPP if there exists a randomized polynomial-time algorithm A such that for every $x \in L$ it holds that $\Pr[A(x,y) = 1] \geq \frac{2}{3}$, and for every $x \notin L$ it holds that $\Pr[A(x) = 0] \geq \frac{2}{3}$. For $\epsilon : \mathbb{N} \to [0,1]$, let BPP_ϵ denote the class of decision problems that can be solved in probabilistic polynomial time with error probability upper-bounded by ϵ. Prove the following two claims:

(a) For every positive polynomial p and $\epsilon(n) = \frac{1}{2} - \frac{1}{p(n)}$, the class BPP_ϵ equals BPP.

(b) For every positive polynomial p and $\epsilon(n) = 2^{-p(n)}$, the class BPP_ϵ equals BPP.

We already proved something similar in class (See Amplication Lemma). This exercise asks you to prove the same using tail bounds. Given an algorithm A, consider an algorithm A' that on input x invokes A on x for $t(|x|)$ times, and decided based on majority as we did in class. For Part (a) $t(n) = O(p(n)^2)$ and apply Chebyshev's Inequality. For Part 2 set $t(n) = O(p(n))$ and apply the Chernoff Bound.

(6) (See Exercise 3.3.2) Let $G(V,E)$ be an undirected graph n vertices which is $(\frac{1}{2}, 2)$-expander. Show that the diameter of the graph is at most $O(\log n)$. The diameter of a graph is at most k if and only if between any two vertices in the graph G, there is a path of length at most k in the graph G.

18.2 Problem Set #2

(1) (See Exercise 4.2.3) A tournament is a directed graph $G(V, E)$ on n vertices where for every pair (i, j), there is either an edge from i to j or from j to i, but not both (it represents real tournaments, where we interpret (i, j) directed edge as player i beats player j. There is no draw and all pairs of players play a game with each other).

A tournament T is said to have k**-championship property** if for any set of k vertices in the tournament, there is some vertex in V that has a directed edge to each of those k vertices.

Can k-Championship property occur in small tournament graphs? For example, for $k = 1$, a tournament will need at least 3 vertices to have the k-Championship property. If $k = 2$, a tournament will need at least 5 vertices to have k-Championship property.

Show that there are tournments of size $O(k^2 2^k)$ having k-Championship property. [Hint : Consider a random tournament. Fix a set S of k vertices and some vertex $v \notin S$. What is the probability that v is the champion in S?]

(2) (See Exercise 4.2.4) Let $G(V, E)$ be a graph. A set of vertices $D \subseteq V$ is called dominating with respect to G if every vertex in $V \setminus D$ is adjacent to a vertex in D. $\delta(G)$, the minimum degree amongst G's vertices, is strictly positive. Then G contains a dominating set of size less than or equal to:

$$\frac{n(1 + \log(1 + \delta))}{1 + \delta}$$

[Hint : Choose a subset $X \subseteq V$ at random (with each vertex in with probability p). Let $Y \subseteq V \setminus X$ having no neighbor in X. Estimate $X \cup Y$.]

(3) (See Exercise 4.3.8) Use expectation method to show that every graph having a matching of size m has a bipartite subgraph with at least $\frac{1}{2}(|E(G)| + m)$ edges. (Hint: how can we choose a random bipartition such that the edges of the matching have their endvertices in opposite parts?)

(4) (See Exercise 4.3.9) Let $t, \ell, d \in \mathbb{N}$ and $t \leq \ell \leq d$. A family of sets $S_1, S_2, \ldots S_m \subseteq [d]$ is said to be a (d, ℓ, t)-design if:

- $\forall i \in [m], |S_i| = \ell$.
- For all $i, j \in [m], i \neq j, |S_i \cap S_j| < t$.

That is, the family is ℓ-uniform (each set of size ℓ) subsets of $[d]$ with intersection sizes at most t. We will have an application soon in the course for such sets. Given ℓ, we would want maximize the number of sets that can be "packed" in - that is maximise m, while keeping t and d to be small. But do such sets exist for all parameters? A typical question that we face in this course, and we use our tools:

(a) Prove that if, $m\binom{\ell}{t}^2 < \binom{d}{t}$, then there exists an (d, ℓ, t)-design $S_1, S_2, \ldots S_m \subseteq [d]$.
Hint : If the sets are chosen at random, then prove that for every $S_1, S_2, \ldots S_{i-1}$:

$$\mathbb{E}_{S_i}\left[\#j < i : |S_i \cap S_j| \geq t\right] < 1$$

(b) Use this to conclude that for every constant $c > 0$, $\ell, m \in \mathbb{N}$, there exists and (d, ℓ, t)-design, $S_1, S_2, \ldots S_m \subseteq [d]$ with $d = O\left(\frac{\ell^2}{t}\right)$ and $t = c \log m$. In particular, setting $m = 2^\ell$, we can fit exponentially many sets of size ℓ in a universe of size $d = O(\ell)$ while keeping the intersections as logarithmically small.

(c) Use method of conditional expectations to derandomize the above to show how to construct designs as in Parts 1 and 2 deterministically in time $\mathsf{poly}(m, d)$.

(5) (See Exercise 5.1.5)

(a) For an $n \times m$ matrix A with Boolean entries and $b \in \{0,1\}^n$, define a function $h_{A,b} : \{0,1\}^m \to \{0,1\}^n$ by $h_{A,b}(x) = (Ax + b) \mod 2$ where the modulo 2 is applied component-wise. Show that

$$H_{m,n} = \{h_{A,b} \mid A \in \{0,1\}^{n \times m} \text{ and } b \in \{0,1\}^n\}$$

is a pairwise independent family of functions. Compare the number of random bits needed to generate a random function in $H_{m,n}$ to the construction that we did in class.

(b) A matrix A is a *Toeplitz matrix* if it is constant on diagonals, i.e., $A_{i+1,j+1} = A_{i,j}$ for all i, j. Show that even if we restrict the family $H_{m,n}$ in Part 1 to only include $h_{A,b}$ for Toeplitz matrices A, we still get a pairwise independent family. How many random bits are needed now?

18.3 Problem Set #3

(1) (See Exercise 6.2.4) Let $G = (V, E)$ be a graph. For every subset S of its vertices V we say, that vertex $v \in V \setminus S$ is a neighbor of S if $E(S, v) \geq 1$, an odd neighbor of S if $E(S, v)$ is odd, a unique neighbor of S if $E(S, v) = 1$. Further, we denote:

$$\begin{aligned} B(S) &= \{v \in V \setminus S \mid v \text{ is a neighbor of } S\} \\ B_{odd}(S) &= \{v \in V \setminus S \mid v \text{ is an odd neighbor of } S\} \\ B_{unique}(S) &= \{v \in V \setminus S \mid v \text{ is a unique neighbor of } S\} \end{aligned}$$

A graph G is said to be an (n, d, α, β)-odd-neibhor (resp. unique-neighbor) expandor if for every $S \subseteq V$, where $|S| \leq \alpha n$, the set B_{odd} (resp. B_{unique}) is of size at least $\beta d|S|$.

(a) Show that every (n, d, α, β) unique-neighbor expander is an (n, d, α, β) odd-neighbor expander.

(b) Show that every (n, d, α, β) odd-neighbor expander is also an (n, d, α, β) boundary expander.

(c) Show that every $(n, d, \alpha, \frac{1}{2} + \frac{\epsilon}{d})$-boundary expander is also a $(n, d, \alpha, \frac{2\epsilon}{d})$-unique-neighbor expander.

(2) (See Exercise 6.2.5) Let $n \in \mathbb{N}$ and $\frac{1}{2} < \alpha < \frac{n-1}{n}$, and $G(V, E)$ by an $(n, d, \frac{1}{2}, \beta)$ edge-expander, then G is also an $(n, d, \alpha, (1 - \alpha)\beta)$ edge expander.

(3) (See Exercise 6.2.6) Through the following steps, show that there exists a family of $(n, d, \frac{1}{2}, \beta)$ edge-expander.

(a) Choose β later. Choose a random d-regular graph on n vertices. Let $S \subseteq V$, with $s = |S| \leq \frac{n}{2}$. Define the random variable $E_S = |E(S, V \setminus S)|$. Let $0 \leq k \leq \beta d|S|$. Prove that if $sd - k$ is an odd number, then, $\Pr[E_S = k] = 0$.

(b) If $sd - k$ is even, then show that

$$\Pr[E_S = k] \leq \left(\frac{3s}{2n}\right)^{ds/4} \qquad \text{Use } \binom{a}{b} \left(\frac{ae}{b}\right)^b \text{ and choose } \beta \text{ such that } \left(\frac{e}{\beta}\right)^{4\beta} \leq \frac{9}{8}$$

(*Hint: Suppose $sd - k$ is even, to calculate probability of k edges leaving the set - select which edges are those k, matching them with any of the $nd - sd$ possible endpoints outside of the set S. Every edge with one endpoint in the subset S has probability roughly s/n that the other endpoint is contained in S as well.*).

(c) Use the previous part to show that :

$$\Pr\left[E_S < \beta ds\right] \leq \left(\frac{5s}{3n}\right)^{ds/4} \qquad \text{Assume } d \geq 140 \text{ and approximate : } \frac{sd}{4} \leq \left(\frac{10}{9}\right)^{sd/4}$$

(d) Prove that the probability that there exists a set of size at most $\frac{n}{2}$, is < 1. Hence conclude the theorem.

(4) (See Exercise 6.7.1) In this problem, we will apply Cheeger's inequality and also show that it is tight.

(a) Let G be a complete graph on n vertices. Show that $h(G) \approx \frac{1}{2}$. Verify Cheeger's inequality.

(b) Let G be a cycle on n vertices. Compute λ_2 and $h(G)$ asymptotically and compare.

(c) Consider the graph G with 2^n vertices labelled with strings in $\{0,1\}^n$. The edges of G are as follows - two vertices to be adjacent if the corresponding strings differ by one bit flip. Show that $\lambda_2 = 1 - \frac{1}{n}$. Hence conclude that $h(G) \geq \frac{1}{2n}$. Derive an upper bound for $h(G)$ and compare.

(5) (See Exercise 6.8.5) Using techniques similar to what we used for mixing lemma, prove the following stronger version of the mixing lemma. Let G be a d-regular graph with n vertices and let $\lambda_2 < 1$ be the second largest eigen value of the normalized adjacency matrix of G. Then, for any $S, T \subseteq [n]$:

$$\left| |E(S,T)| - \frac{d|S||T|}{n} \right| \leq d\lambda_2 \left(\sqrt{|S||T| \left(1 - \frac{|S|}{n}\right) \left(1 - \frac{|T|}{n})\right)} \right)$$

18.4 Problem Set #4

(1) (See Exercise 10.1.3) This question discusses a construction of an explicit $\frac{1}{2}$-hitting disperser from an explicit pairwise independent hash family. Given parameters $V = \{1, 2, \ldots m\}$ and $\epsilon > 0$, let D be a set of size more than $\lceil \frac{1}{\epsilon} \rceil$. Let H be a pairwise independent hash family that has functions from D to V such that the size of H is polynomial in $|D|/|V|$. Define the following disperser (U, V, E), with $U = H$ and let

$$E = \{(h, h(x)) \mid h \in U, x \in D\}$$

That is, each vertex of U corresponds to a hash function, and each outgoing edge from such a vertex corresponds to applying this hash function to a particular value in D. Show that what we constructed is a $\frac{1}{2}$-hitting disperser with $|U|$ polynomially bounded in $|V|/\epsilon$ with degree $O(\frac{1}{\epsilon})$ and threshold $\epsilon|U|$.

Hint : It suffices to show that if we take any subset V' of V of size at least $\frac{|V|}{2}$ and a random vertex $h \in U$, the probability that there is an edge between h and V' is more than $1 - \epsilon$. Show that this sufficient and then prove the same. In the probability calculation, Chebychev inequality and part (c) of a question from midsem will come handy, which you can use without proof for the purpose of this question.

(2) (See Exercise 11.5.3) If D is an ϵ-biased distribution over $\{0,1\}^m$, then as a vector $D \in \mathbb{R}^{2^m}$:

$$\frac{1}{|\mathsf{supp}(D)|} \leq \|D\|_2^2 \leq \epsilon^2 + \frac{1}{2^m}$$

(3) (See Exercise 11.5.4) Let $G : \{0,1\}^{k \log m} \to \{0,1\}$ be the generator that we described such that $G\left(U_{k \log m}\right)$ outputs a k-wise independent distribution. (See the explicit construction of k-wise independent distribution). If we replace the input to G with a small bias distribution of $\epsilon' = \frac{\epsilon}{2^k}$, then the output of G is ϵ-close to being k-wise independent. Thus, conclude that, there is a generator for almost k-wise independent distributions with seed length $O\left(\log \log m + k + \log(1/\epsilon)\right)$.

(4) (See Exercise 11.6.6) A k universal set $S \subseteq \{0,1\}^n$ has the property that the projection of S onto any k indexes contains all 2^k possible patterns. Use the previous question to construct k-universal sets of size $\left(2^k \log n\right)^{O(1)}$.

18.5 Problem Set #5

(1) (See Exercise 13.1.6) A classic data structure problem is to map a large universe U to a (possibly) smaller domain Σ. We call a hash function $h : U \to \Sigma$ as *good* if $h(x) \neq h(y)$ for all $x \neq y \in U$ (if not, we call it a collision). Indeed, this requires that $|\Sigma| \geq |U|$ and hence is impossible. An interesting variant is to relax on the collision requirement - define a family of hash functions and then argue that for any pair of elements from the universe U, the probability of collision is small for a hash function chosen randomly from the family. More formally, a family of functions from U to M, denoted by $H = \{h_1, h_2, \ldots h_n\}$ is a ϵ-good hash family if for every $x \neq y \in U$, $\Pr_{h \in H}[h(x) = h(y)] \leq \epsilon$.

The task is to construct explicitly, ϵ-good families of hash functions of small size. In this problem, we will use codes with large distance to construct ϵ-good hash families. In fact, we show that the two tasks are equivalent.

If $|\Sigma| = q$, given an $(n, k, d)_q$ code, we define a family of hash functions as follows:

$$H_C = \left\{ h_i : \Sigma^k \to \Sigma \mid 1 \leq i \leq n, \forall x \in \Sigma^k, h_i(x) \stackrel{\text{def}}{=} C(x)_i \right\}$$

(a) Prove that if C is an $(n, k, \delta n)_q$ code, then H_C is $(1 - \delta)$-good. Comment on the size of the family.

(b) Defining the code from the hash family appropriately, show that if we have a hash family which is ϵ-good, then we also get a code from it which is $(n, k, (1 - \epsilon)n)_q$.

(2) (See Exercise 13.3.2) Let C_1 be an $[n_1, k_1, d_1]_2$ binary linear code, and C_2 be an $[n_2, k_2, d_2]$ binary linear code. Let $C \subseteq \mathbb{F}_2^{n_1 \times n_2}$ be the subset of $n_1 \times n_2$ matrices whose columns belong to C_1 and whose rows belong to C_2. Show that C is an $[n_1 n_2, k_1 k_2, d_1 d_2]_2$ binary linear code.

(3) (See Exercise 13.3.9)

For integers $1 \leq k \leq n$, call a subset $S \subseteq 0, 1^n$ to be k-wise independent if for every $1 \leq i_1 < i_2 < \cdots < i_k \leq n$ and $a \in 0, 1^k$

$$\Pr_{x \in S}[x_{i_1} = a_1 \wedge x_{i_2} = a_2 \wedge \cdots \wedge x_{i_k} = a_k] = \frac{1}{2^k}$$

where the probability is over an element x chosen uniformly at random from S. In this problem, you will see how codes can be used to construct k-wise independent sets of small size. Recall the constructions that we have seen earlier.

(a) Let $H \in \mathbb{F}_2^{t \times n}$ be the parity check matrix of an $[n, n - t, d]_2$ binary linear code of distance $d > k + 1$. Define:

$$S = \left\{ x^T H \mid x \in \mathbb{F}_2^t \right\}$$

Prove that S is a k-wise independent set of size 2^t.

(b) Using the above, show how one can construct a k-wise independent subset of $\{0, 1\}^n$ of size at most $(2n)^{(k/2)}$.

(4) (See Exercise 13.5.1) Consider a game involving n people in a room, each of whom is given a red/white hat chosen uniformly and independently at random. Each person can see the hat color of all other people, but not their own. They are asked to guess their own hat color which they can decline or make a guess (simultaneously). They either win collectively, or lose collectively. Once the guessing is done, they win if no body makes a wrong guess and at least one person makes the correct guess. They cannot interact during the game, but can have some predetermined strategy among themselves.

A trivial strategy is to make only one person guess and everyone else decline. This gives a probability of $\frac{1}{2}$. In this problem, our goal is to come up with a much better strategy. The problem presents an interesting application of Hamming Codes.

A directed graph G is a subgraph of the n-dimensional hypercube if its vertex set is $\{0,1\}^n$ and if (u,v) is an edge in G, then $\Delta(u,v) = 1$ (hamming distance).

For a graph G, let $K(G)$ be the number of vertices of G with in-degree at least one, and out-degree zero.

(a) Show that: $\Pr\{\text{Winning Hat Game}\} = \max_G \frac{K(G)}{2^n}$ (Hint : Establish a bijection between strategies and subgraphs. Think of the endpoints of an edge as two possible views for a player.)

(b) Using the fact that the out-degree of any vertex is at most n, show that, for any directed subgraph G of the n-dimensional hypercube :

$$\frac{K(G)}{2^n} \leq \frac{n}{n+1}$$

(c) Use hamming codes to show that if $n = 2^\ell - 1$ then there exists a directed subgraph of G of the n-dimensional hypercube with

$$\frac{K(G)}{2^n} = \frac{n}{n+1}$$

Bibliography

[Alon et al., 1992] Alon, N., Goldreich, O., Hastad, J., and Peralta, R. (1992). Simple constructions of almost k-wise independent random variables. *Random Structures & Algorithms*, 3(3):289–304.

[Beck, 1978] Beck, J. (1978). On 3-chromatic hypergraphs. *Discrete Mathematics*, 24(2):127 – 137.

[Ben-Aroya and Ta-Shma, 2009] Ben-Aroya, A. and Ta-Shma, A. (2009). Constructing small-bias sets from algebraic-geometric codes. In *2009 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 191–197.

[Bilu and Linial, 2006] Bilu, Y. and Linial, N. (2006). Lifts, discrepancy and nearly optimal spectral gap. *Combinatorica*, 26(5):495–519.

[Erdos, 1963] Erdos, P. (1963). On a Combinatorial Problem. *Nordisk Matematisk Tidskrift*, 11(1):5–10.

[Gabber and Galil, 1981] Gabber, O. and Galil, Z. (1981). Explicit constructions of linear-sized superconcentrators. *J. Comput. System Sci.*, 22(3):407–420. Special issued dedicated to Michael Machtey.

[Goemans and Williamson, 1995] Goemans, M. X. and Williamson, D. P. (1995). Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42(6):1115–1145.

[Khot, 2010] Khot, S. (2010). On the unique games conjecture (invited survey). pages 99–121.

[Khot et al., 2007] Khot, S., Kindler, G., Mossel, E., and O'Donnell, R. (2007). Optimal inapproximability results for max-cut and other 2-variable csps? *SIAM J. Comput.*, 37(1):319–357.

[Lubotzky et al., 1988] Lubotzky, A., Phillips, R., and Sarnak, P. (1988). Ramanujan graphs. *Combinatorica*, 8:261–277.

[Margulis, 1973] Margulis, G. A. (1973). Explicit constructions of expanders. *Problemy Peredači Informacii*, 9(4):71–80.

[Murty, 2003] Murty, M. R. (2003). Ramanujan graphs. *J. Ramanujan Math. Soc*, 18.

[Naor and Naor, 1990] Naor, J. and Naor, M. (1990). Small-bias probability spaces: Efficient constructions and applications. In *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing*, STOC '90, page 213–223, New York, NY, USA. Association for Computing Machinery.

[Naor and Naor, 1993] Naor, J. S. and Naor, M. (1993). Small-bias probability spaces: Efficient constructions and applications. *SIAM J. Comput.*, 22(4):838–856.

[Radhakrishnan and Srinivasan, 2000] Radhakrishnan, J. and Srinivasan, A. (2000). Improved bounds and algorithms for hypergraph 2-coloring. *Random Structures & Algorithms*, 16(1):4–32.

[Reingold, 2008] Reingold, O. (2008). Undirected connectivity in log-space. *J. ACM*, 55(4):17:1–17:24.

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

[Schmidt, 1964] Schmidt, W. M. (1964). Ein kombinatorisches Problem von P. Erdos und A. Hajnal. *Acta Mathematica Academiae Scientiarum Hungarica*, 15(3):373–374.

[Ta-Shma, 2017] Ta-Shma, A. (2017). Explicit, almost optimal, epsilon-balanced codes. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, page 238–251, New York, NY, USA. Association for Computing Machinery.