CS5275 Lecture 1: Basic Tools

Jonathan Scarlett

February 21, 2025

Useful references:

- "Methods of Proof" blog posts: `https://www.jeremykun.com/primers/`
- CMU lecture on Big-O: `https://www.youtube.com/watch?v=_gKb855_3bk`
- CMU lecture on online resources: `https://www.youtube.com/watch?v=qP4XEZ54eSc`
- Inequality sheet: `https://www.lkozma.net/inequalities_cheat_sheet/ineq.pdf`

Note on examination: Nothing in this lecture will be examined *directly*, but many of the tools/concepts will be used when studying other topics.

1 General Background

This lecture gives some very broad background, much of which might already be familiar. After this one, the subsequent lectures will be more focused and specific.

The main assumed CS background for this course is:

- Algorithms and data structures
- Some exposure to tools like optimization, graphs, etc.
- Some exposure to computational complexity

The main assumed mathematical background for this course is:

- Probability *(see background document)*
- Linear algebra *(see background document)*
- Basic calculus
- Some exposure to tools like Taylor expansion, limits, binomial coefficients, etc.

2 Proof Techniques

Some of the most common proof techniques are outlined as follows with one example each.

Proof technique 1: Direct proof

- Idea: Just find a sequence of steps that leads directly from the assumptions to the desired statement
- Often this is done via a (possibly long) sequence of steps, e.g.:
  - To show $A \implies B$, show $A \implies S_1$, then $S_1 \implies S_2$, etc., up to $S_{k-1} \implies S_k$ and $S_k \implies B$
  - To show $a \leq b$, just show $a \leq \ldots \leq b$ (or similarly for $a = b$ or $a \geq b$)
- Example: Consider the claim "The product of 3 consecutive positive integers is always a multiple of 6." We can simply combine the following direct observations:
  - Fix $n \geq 1$. Since one (or two) of $n, n+1, n+2$ must be even, we find that $n(n+1)(n+2)$ is an even number (i.e., a multiple of 2).
  - Similarly, exactly one of $n, n+1, n+2$ is a multiple of 3, so $n(n+1)(n+2)$ is a multiple of 3.
  - Since $n(n+1)(n+2)$ is a multiple of both 2 and 3, it is a multiple of 6.

Proof technique 2: Contrapositive

- Idea: To show something of the form $A \implies B$, show the equivalent statement $B^c \implies A^c$, where $(\cdot)^c$ means the complement ("not").
- Example:
  - Consider the claim "$\sqrt{n}$ is irrational for any non-square positive integer n".
  - We can re-word this into "$A \times B$" form as follows: "For any positive integer n, if n is not a square number, then $\sqrt{n}$ is irrational."
  - The contrapositive statement is "For any positive integer n, if $\sqrt{n}$ is rational, then n is a square number". This turns out to be more convenient to prove.
  - The details:
    - If $\sqrt{n}$ is rational, then there must exist integers a, b such that $\sqrt{n} = \frac{a}{b}$, or equivalently $nb^2 = a^2$.
    - Now factor a, b, n into a product of primes (uniquely). Since both a^2 and b^2 have every prime appearing an even number of times, the same must be true of n. Thus, n is a square number.

Proof technique 3: Contradiction

- Idea: To show that a claim is true, start by assuming that it is false, and show that this leads to a mathematical contradiction.
  - Note: This is not the same as the contrapositive technique (in which nothing contradictory is assumed), and it can be applied to other statements beyond those of the form "$A \implies B$"

- Example:
  - Consider proving the claim "There are infinitely many prime numbers" (many proofs are known!)
  - Proceed with a proof by contradiction, assuming that there are only finitely many prime numbers. That is, there exists a finite N such that the prime numbers are $p_1, \ldots, p_N$.
  - Define $M = p_1 \times p_2 \times \ldots \times p_M$, and consider the integer $M+1$. Since every integer can be written as a product of primes, there must exist some p_i that is a factor of $M+1$.
  - But p_i is also trivially a factor of M.
  - Being a factor of both M and $M+1$ is impossible, since $p_i \geq 2$ – a contradiction.

Proof technique 4: Induction

- Idea: To prove that a claim C_n holds for all integers $n \geq n_0$ (where n_0 is typically 0 or 1), first prove C_{n_0} (base case) and then prove that $C_n \implies C_{n+1}$ (induction step).
- Example: In the Towers of Hanoi problem, there are n disks stacked from largest (bottom) to smallest (top):

  The goal is to move all the disks to the right-hand side pole, but only one ring can be moved at a time, and *a larger disk can never be above a smaller disk.*
- Claim: This problem can be solved in $2^n - 1$ moves (in fact this is optimal, but we won't prove that)
- Proof:
  - The base case is $n = 1$, in which case $2^n - 1 = 1$ and the result is trivial.
  - For the induction step, we need to show that if $2^n - 1$ moves suffices with n disks, then $2^{n+1} - 1$ moves suffices with $n+1$ disks.
  - To see this, we perform 3 steps:
    - Use the n-disk method to move the n smallest disks to the middle pole ($2^n - 1$ moves)
    - Move the largest disk to the right pole (1 move)
    - Use the n-disk method to move the n smallest disks to the right pole ($2^n - 1$ moves)

    The total number of moves is $2(2^n - 1) + 1 = 2^{n+1} - 1$, as desired.

Other proof techniques:

- Proof by cases (a special case being proof by exhaustion)
- Proof by example for "there exists..." statements (or by counter-example for "not all..." statements)
- The probabilistic method (which we will cover later)

- Visual proofs
- Proof of equality by showing $\geq$ and $\leq$ separately
- Other logic-based proofs (e.g., to show $A \implies (B_1 \text{ or } B_2)$, show that when A is true, $B_1^c \implies B_2$)

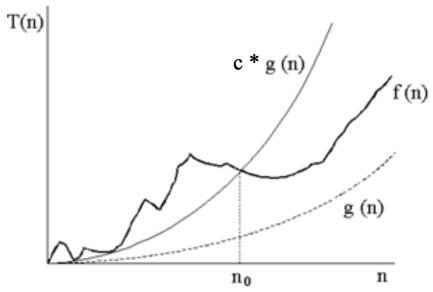
Errors in proofs

- There are endless ways that proofs can contain errors (even when the final result is correct), and they sometimes enter in very subtle or unexpected ways.
- Example 1: It is easy to accidentally take $(a^b)^c$ and a^{bc} to be the same. This is not always true!
  - Particular care should be taken with complex numbers, e.g.:, for any $\theta \in \mathbb{R}$ we know that $e^{i\theta} = \cos\theta + i\sin\theta$, whereas
    $$(e^{2\pi i})^{\frac{\theta}{2\pi}} = 1^{\frac{\theta}{2\pi}} = 1$$
    which is clearly different from $\sin\theta + i\cos\theta$ except for some very specific θ values.
  - Even without complex numbers, $a < 0$ can cause similar issues (e.g., $a = -1$, $b = 2$, $c = \frac{1}{2}$)
  - On the other hand, if a, b, c are all real-valued and $a > 0$, then indeed $(a^b)^c = a^{bc}$.
  - (Note: This problem could be alleviated by treating both ± 1 as square roots of 1, or more generally taking $e^{2\pi i/n}$ to be an n-th root of 1 for every $i = 1, \ldots, n$.)
- Example 2: Consider a setup where we repeatedly roll a 6-sided die until we observe a 6. What's the average number of rolls (including the roll that gave 6) conditioned on only seeing even numbers?
  - Incorrect intuition: 3, because without even numbers we have $\{2, 4, 6\}$ being equally likely. *(This intuition would be correct if we were to condition on an **infinite** sequence of rolls containing no even numbers.)*
  - Correct intuition: 1.5, because we can think of this as repeatedly rolling until we see anything except $\{2, 4\}$ (and if it's odd, we reset the experiment). Hence, the stopping time has a Geometric(2/3) distribution, whose mean is 1.5.
  - (This isn't a rigorous argument, but a formal proof via the definition of conditional probability is also possible, and gives 1.5.)

3 Asymptotic Notation

Big-O notation:

- For two real-valued sequences f_n and g_n indexed by $n \in \{1, 2, 3, \ldots\}$, we say that $f_n = O(g_n)$ if there exist constants $C > 0$ and $n_0 > 0$ such that $|f_n| \leq Cg_n$ for all $n \geq n_0$.
  - More concisely: f_n is upper bounded by a constant times g_n when n is large enough
  - An illustration (from UWash CSE373 slides):
- Notes on notation:



  - Technically $O(\cdot)$ is a *set of sequences*, so notation like $f_n \in O(g_n)$ would be more precise. But more often we just write $f_n = O(g_n)$.
  - We also tend to combine other operations with $O(\cdot)$, e.g., writing things like $f_n = n^2 + e^{O(\sqrt{\log n})}$. It can get even more confusing when there are multiple variables involved (more on this below).

- Other limits: We have introduced $O(\cdot)$ with respect to asymptotics in the limit of a parameter $n \to \infty$, and we will mostly focus on this below. But the notation is also used with respect to other limits, such as some $\epsilon \to 0$ (e.g., $f(\epsilon) = 1 + \epsilon + O(\epsilon^2)$). The context should always make the precise limit clear, otherwise it should be stated explicitly.

Standard variations:

- (Ω) The statement $f_n = \Omega(g_n)$ is equivalent to $g_n = O(f_n)$. That is, f_n is lower bounded by a constant times g_n (when n is large enough).
- (Θ) The statement $f_n = \Theta(g_n)$ is equivalent to having both $f_n = O(g_n)$ and $g_n = O(f_n)$. That is, f_n and g_n coincide to within a constant factor.
- (o) The statement $f_n = o(g_n)$ is equivalent to $\lim_{n\to\infty} \frac{f_n}{g_n} = 0$. That is, f_n is strictly smaller than g_n asymptotically.
- (ω) The statement $f_n = \omega(g_n)$ is equivalent to $g_n = o(f_n)$. That is, f_n is strictly larger than g_n asymptotically.

Variations omitting log factors:

- The statement $f_n = \widetilde{O}(g_n)$ is equivalent to there existing some constant c such that $f_n = O(g_n(\log g_n)^c)$. This notion is useful when the goal is to get the "leading terms" while treating log factors as less significant.
  - For example, $n^2 \log^3 n = \widetilde{O}(n^2)$, and $2^n n^2 \log n = \widetilde{O}(2^n)$.
  - Statements like "$f_n = O(g_n(\log g_n)^c)$ for some c" are also often written as $f_n = O(g_n \text{poly}(\log g_n))$
- Note: While you can think of $\widetilde{O}(g_n)$ as "like $O(g_n)$ but ignoring log factors", it should always be remembered that this means factors that are *logarithmic in g_n, not necessarily in n*. For example:
  - $\widetilde{O}(n)$ and $\widetilde{O}(n^2)$ hide $(\log n)^c$ factors;
  - $\widetilde{O}(2^n)$ or $\widetilde{O}(e^n)$ hides n^c factors, because $\log_2(2^n) = \log_e(e^n) = n$.
  - $\widetilde{O}(\log n)$ is only allowed to hide $(\log \log n)^c$ factors.

– $\widetilde{O}(1)$ makes no sense!

- We can similarly consider $\widetilde{\Omega}$ and $\widetilde{\Theta}$ (whereas $\widetilde{o}$ and $\widetilde{\omega}$ are not standard notions.)

Examples:

- We tend to state algorithm runtimes in $O(\cdot)$ notation, especially when the precise constants depend on implementation details (e.g., $O(n \log n)$ time for sorting).
- Big-O notation is often closely tied to Taylor expansions. For example, we can write $\sqrt{1+x} = 1+O(x)$ as $x \to 0$, or include higher-order terms like $\sqrt{1+x} = 1 + \frac{x}{2} + O(x^2)$ as $x \to 0$.
  - To highlight the importance of different limits, note that the limit $x \to \infty$ gives the completely different behavior of $\sqrt{1+x} = O(\sqrt{x})$.
  - To get a more precise expression as $x \to \infty$ we could also write $\sqrt{1+x} = \sqrt{x} \cdot \sqrt{1+1/x}$ and then use the above findings, e.g.,

$$\sqrt{1+x} = \sqrt{x} \cdot \left(1 + \frac{1}{2x} + O\Big(\frac{1}{x^2}\Big)\right) = \sqrt{x} + \frac{1}{2\sqrt{x}} + O(x^{-3/2}).$$

  - Note: Wolfram Alpha is a useful tool for getting Taylor expansions without doing them by hand
- *Stirling's approximation* is often written using asymptotic notation, e.g.,

$$\log(n!) = n \log n - n + O(\log n).$$

  This leads to similar sorts of statements for the binomial coefficients $\binom{n}{k} = \frac{n!}{k!(n-k)!}$, e.g.:
  - (Coarse behavior) $\log \binom{n}{k} = O\big(k \log \frac{n}{k}\big)$
  - (Precise behavior when $k = \alpha n$ with $\alpha \in (0,1)$) $\log \binom{n}{k} = nH_2(\alpha)(1+o(1))$ where $H_2(\alpha) = \alpha \log \frac{1}{\alpha} + (1-\alpha) \log \frac{1}{1-\alpha}$ is the binary entropy function.
  - (Precise behavior when $k = o(n)$) $\log \binom{n}{k} = \big(k \log \frac{n}{k}\big)(1+o(1))$.
- We often use $o(\cdot)$ notation to hide "lower-order terms". For example, suppose that we want to show that some probability is exponentially decaying in n, and we get a bound of the form

$$\mathbb{P}[\text{event}] \le e^{-nC}\left(1 + O\Big(\frac{1}{n}\Big)\right) + e^{-n \log n}.$$

  The second term is $e^{-\omega(n)}$ (i.e., it decays faster than exponential). So we might summarize the above by the simpler statement

$$\mathbb{P}[\text{event}] \le e^{-nC}(1+o(1)).$$

  In contrast, note that if we started with the expression $e^{-nC(1+o(1))}$, we could *not* simplify it to $e^{-nC}(1+o(1))$. This is because $e^{-nC(1+o(1))} = e^{-nC} \times e^{o(n)}$, but $e^{o(n)}$ could still by very large or very small (e.g., n^{100} or n^{-100}, or something even more significant like $e^{\sqrt{n}}$ or $e^{-\sqrt{n}}$).

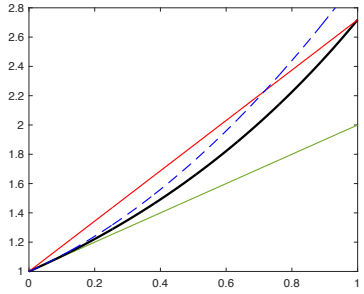
Cautionary notes:

- As evidenced by the last example above, we should be careful with non-linear functions, e.g., if $y = O(x)$ then we can say that $y^2 = O(x^2)$ but we *cannot* say that $e^y = O(e^x)$ (i.e., $e^{O(x)}$ and $O(e^x)$ are not equivalent).
  - Along similar lines, another common mistake is incorrect cancellations, e.g., if $f_n = e^{n+o(n)}$ and $g_n = e^{n+o(n)}$ then we have $\frac{f_n}{g_n} = e^{o(n)}$, but we cannot say something similar about $f_n - g_n$ (e.g., consider $f_n = 2e^n = e^{n+\ln 2}$ and $g_n = e^n$).
- If limits are being taken with respect to multiple variables, then $O(\cdot)$ notation could become unclear, because limits taken in different orders can give different results. I tend to think of it as follows: To make a statement like $f(m, \delta, \epsilon) = O\big(\frac{m}{\epsilon^2} \log \frac{1}{\delta}\big)$ with $m \to \infty$, $\delta \to 0$, and $\epsilon \to 0$, it should be the case that *any* sequence $\{(m_n, \delta_n, \epsilon_n)\}_{n=1}^{\infty}$ with $m_n \to \infty$, $\delta_n \to 0$ and $\epsilon_n \to 0$ should satisfy $f(m_n, \delta_n, \epsilon_n) = O\big(\frac{m_n}{\epsilon_n^2} \log \frac{1}{\delta_n}\big)$ as $n \to \infty$.

4 Inequalities

Inequalities are ubiquitous in theoretical computer science; some common ones are outlined as follows (see the sheet linked on p1 for many more):

- Basic inequalities, e.g., $1 + x \le e^x$ or equivalently $\log(1 + x) \le x$
  - Even when there is no hope of analogous reverse inequalities that hold for all x (e.g., $1 + cx$ can never be a universal upper bound on e^x, no matter how large c is), we can often get those for a *restricted range of* x:



  - The bold curve is e^x, the solid straight lines are $1 + x$ and $1 + (e - 1)x$, and the dashed curve is $1 + x + x^2$. Restricted to $x \in [0, 1]$, the latter two are upper bounds on e^x.
- Cauchy-Schwarz inequality: $|\langle \mathbf{u}, \mathbf{v} \rangle| \le \|\mathbf{u}\| \cdot \|\mathbf{v}\|$
  - Generalizes to Hölder's inequality: $|\langle \mathbf{u}, \mathbf{v} \rangle| \le \|\mathbf{u}\|_p \|\mathbf{v}\|_q$ when $p, q \ge 1$ satisfy $\frac{1}{p} + \frac{1}{q} = 1$ (e.g., $|\langle \mathbf{u}, \mathbf{v} \rangle| \le \|\mathbf{u}\|_1 \|\mathbf{v}\|_\infty$). Here $\|\mathbf{u}\|_p = \big(\sum_i |u_i|^p\big)^{1/p}$ is the "p-norm", and $\|\mathbf{u}\|_\infty = \max_i |u_i|$.
  - Both generalize beyond real-valued vectors, e.g., $\mathbb{E}[|XY|] \le \big(\mathbb{E}[|X|^p]\big)^{1/p} \big(\mathbb{E}[|Y|^q]\big)^{1/q}$.
- Triangle inequality: $|a + b| \le |a| + |b|$, or for vectors $\|\mathbf{u} + \mathbf{v}\| \le \|\mathbf{u}\| + \|\mathbf{v}\|$.
- Bounds on binomial coefficient, e.g.,

$$\Big(\frac{n}{k}\Big)^k \le \binom{n}{k} \le \Big(\frac{ne}{k}\Big)^k$$

(This is less precise than Stirling's approximation, but convenient due to being non-asymptotic.)

- Union bound (AKA Boole's inequality):

$$\mathbb{P}\Big[\bigcup_{i=1}^{N} A_i\Big] \leq \sum_{i=1}^{n} \mathbb{P}[A_i].$$

- Arithmetic mean vs. geometric mean: $\left(\prod_{i=1}^{n} x_i\right)^{1/n} \leq \frac{1}{n}\sum_{i=1}^{n} x_i$.

- Inequalities between norms, e.g., for $\mathbf{u} \in \mathbb{R}^n$:

$$\|\mathbf{u}\|_2 \leq \|\mathbf{u}\|_1 \leq \sqrt{n}\|\mathbf{u}\|_2$$
$$\|\mathbf{u}\|_\infty \leq \|\mathbf{u}\|_2 \leq \sqrt{n}\|\mathbf{u}\|_\infty.$$

  Note that $\|\mathbf{u}\|_1 = \sum_i |u_i|$ and $\|\mathbf{u}\|_\infty = \max_i |u_i|$, and $\|\mathbf{u}\|_2 = \sqrt{\sum_i u_i^2}$ is the regular Euclidean norm.

- We will not need many (if any) matrix inequalities in this course, but for an extensive list, search "Matrix Cookbook" in Google.

- Jensen's inequality: $f(\mathbb{E}[X]) \leq \mathbb{E}[f(X)]$ for convex f (see the later lecture on convex optimization)

- Probabilistic bounds and concentration inequalities (next lecture): Markov, Chebyshev, Bernstein, Chernoff, Hoeffding, McDiarmid, etc.

5 Note on Computational Complexity

- This course will be fairly light on computational complexity (consider taking CS5230!), but here we pause to mention a model that is widely used *implicitly* and is worth seeing more explicitly at least once (though we will avoid giving a very formal description).

- Motivating example:

  - Suppose that we have a sorted list of n numbers, and let's say each of them as a positive integer in the range $\{1, 2, \ldots, n^{10}\}$ (the number 10 here is arbitrary). What's the computational complexity of finding the first number to exceed a specified threshold γ? (e.g., in the list $\{1, 5, 9, 10, 53, 94, 1024, 1999\}$, find the first number greater than 100)
  - This can be solved using *binary search* – check the middle element, then move to the left half or right half depending on whether it exceeds γ, and continue recursively.
  - We recurse $O(\log n)$ times before narrowing down to a single element, so it's natural to say that the computational complexity is $O(\log n)$.
  - However, this raises the following concern: *Representing an integer in* $\{1, 2, \ldots, n^{10}\}$ *requires* $O(\log n)$ *bits.* If we read those bits one-by-one to compare two numbers, and each bit takes a constant amount of time to read, then each comparison takes $O(\log n)$ time and thus the total time is $O((\log n)^2)$.

- The *word RAM model* overcomes this ambiguity by assuming that reading an *entire* number (and performing "basic" operations on them) in fact takes constant time, in which case the complexity is indeed $O(\log n)$ in the above example.
  - Examples of basic operations: Comparison, addition, bit-wise AND/OR/NOT, etc. (there is a formal class called AC^0 containing these)
  - Less basic operations like multiplication and division are also often considered to take constant time, though not always (as it is a bit more questionable than addition).
- It could be argued that it's strange to assume "words" can be stored and read in a manner that grows with the input size n, but this model is quite well-aligned with how modern computers work and the fact that 64-bit words suffice for most practical purposes. However, there are some subtle issues:
  - The word length w should be thought of being at least $\log n$ (so that we can at least index integers from 1 to n) but at most $O(\log n)$ (meaning each number can take one of at most $\text{poly}(n)$ values, like n^{10} in the example above). A word size $w = \omega(\log n)$ tends to be "disallowed", as this would mean being able to index super-polynomially many memory locations in constant time.
  - Strictly speaking a real-valued number requires infinitely many bits to store exactly, so extra care may be needed for algorithms that (conceptually) operate on real numbers. This can usually be ignored when the real numbers are being used in "reasonable" ways (e.g., we don't cheat by hiding information in the very insignificant bits of the real number).
  - Roughly speaking, if the algorithm works on "real" numbers, then they should be "sufficiently accurately" represented using $O(\log n)$ bits. This is usually a reasonable assumption in practice.

6 Useful Resources

For basic mathematical checks, calculations, etc., some useful resources are as follows:

- Wikipedia is a perfectly fine starting point for a concept you've never heard of (e.g., "Stirling numbers of the second kind" or "BPP complexity class"), though it might sometimes be lacking in detail and/or require more scrutiny compared to other sources.
- Wolfram Alpha (online) for basic integrals, Taylor expansions, limits, etc.
- Wolfram Mathematica (software) for more sophisticated calculations along similar lines
  - Maple is along similar lines.
  - MATLAB is common but usually used more for numerical computations.
- Inverse Symbolic Calculator (online) for looking up constants (e.g., try 0.57721566490), or similarly OEIS for integer sequences (e.g., try 1,1,2,5,14,42)
- Optimization libraries / packages (e.g., Gurobi, CPLEX, Yalmip, CVX)
- MathOverflow/StackOverflow for endless Q&A posts (e.g., try typing into Google "StackOverflow difference between NP and co-NP")
- If you find yourself having to search through research papers, clever exploration could save you time:

- If the paper's result seems too complicated or "over-the-top" for your purposes, check the Related Work section (or reference list) for possible simplified versions done in earlier works
- Conversely, if the paper's result isn't strong enough for your purposes, try searching the paper on Google Scholar, then clicking "Cited By", possibly followed by a search with "Search within citing articles" ticked.

See https://www.youtube.com/watch?v=qP4XEZ54eSc for a "Street Fighting Mathematics" lecture along these lines.

7 CS5275 Topics

The main topics we will cover are as follows:

- Concentration inequalities
- The probabilistic method
- Convexity and convex optimization
- Submodularity and discrete optimization
- Multiplicative weights algorithms
- Fourier transform
- Information theory
- Error-correcting codes
- Expander graphs
- Communication complexity

We will also have a lecture touching very briefly on "topics we didn't get to cover fully":

- Distance measures (particularly between probability distributions)
- Matrix decompositions
- Further probabilistic limit theorems
- Computational complexity
- Constraint satisfaction
- Sketching and streaming
- Hashing
- Derandomization and psuedorandomness
- Random graph theory
- Spectral graph theory

- Other graph algorithms/tools (e.g., sparsifiers, bouded treewidth)
- Cryptography

CS5275 Lecture 2: Concentration Inequalities

Jonathan Scarlett

January 20, 2025

Useful references:

- Blog post by Jeremy Kun[1]
- First section of Boucheron *et al.*'s "Concentration Inequalities" notes[2]
- Chapter 2 of Vershynin's book "High Dimensional Probability"
- CS-style course notes: Chapter 2 of USyd course `https://ccanonne.github.io/teaching/COMPx270`
- CS-style textbooks: "Concentration of Measure for the Analysis of Randomized Algorithms" (Panconesi/Dubhashi) and "Randomized Algorithms" (Motwani/Raghavan)

Categorization of material:

- Core material: Sections 1–4 and Examples 1–4 of Section 6
- Extra material: Section 5, rest of Section 6, Section 7

(Exam will strongly focus on "Core". Take-home assessments may occasionally require consulting "Extra".)

1 Introduction

- Given a random variable Y, how "concentrated" is Y (e.g., around its mean)? We will particularly be interested in the case where $Y = f(X_1, \dots, X_n)$ is a function of n independent random variables $X_1, \dots, X_n$, such as the empirical average $Y = \frac{1}{n}\sum_{i=1}^n X_i$.
- Let $Y = Y_n$ to make explicit that Y depends on n. Roughly, a concentration inequality is an inequality stating that there exists a deterministic value m such that

$$\mathbb{P}[|Y_n - m| > t] \le \text{TailBound}(n, t)$$

  where $\text{TailBound}(n, t)$ ideally decreases to 0 rapidly as n increases.

  - Typically $m = \mathbb{E}[Y_n]$ (other choices may include $m = \text{median}(Y_n)$ or $m = 0$), and often $\text{TailBound}(n, t)$ decreases exponentially, such as $\text{TailBound}(n, t) \sim e^{-cnt^2}$ for some $c > 0$.
  - Such results are useful because they tell us that *the behavior of Y_n becomes more and more predictable as n increases*; namely, we know that Y_n will be very close to m with high probability.

[1] `http://jeremykun.com/2013/04/15/probabilistic-bounds-a-primer/`
[2] `http://www.econ.upf.edu/~lugosi/mlss_conc.pdf`

  - In statistics, Y may be a quantity being estimated from data. In computer science, Y can represent the outcome of a randomized algorithm. There are many other applications in information theory, statistical learning theory, statistical physics, random graph theory, random matrix theory, etc.

- Simple example: Suppose $Y_n = \frac{1}{n}\sum_{i=1}^n X_i$, where the X_i are i.i.d. with mean μ and variance σ^2.
  - **Law of Large Numbers:** $\mathbb{P}[|Y_n - \mu| > \epsilon] \to 0$ as $n \to \infty$.
  - **Central Limit Theorem:** $\mathbb{P}\big[|Y_n - \mu| > \frac{\alpha}{\sqrt{n}}\big] \to 2\Phi\big(-\frac{\alpha}{\sigma}\big)$ as $n \to \infty$, where Φ is the standard normal CDF.
  - **Large Deviations:** Under some technical assumptions, $\mathbb{P}[|Y_n - \mu| > \epsilon] \le e^{-n\cdot\psi(\epsilon)}$ for some $\psi(\epsilon) > 0$. This type of result is the focus of this lecture.
  - **Moderate Deviations:** Decay rate of $\mathbb{P}[|Y_n - \mu| > \epsilon_n]$ when $\epsilon_n \to 0$ sufficiently slowly so that $\epsilon_n\sqrt{n} \to \infty$.

- In many applications, we want the bounds to be *non-asymptotic* (i.e., holding for any n, as opposed to only in the limit $n \to \infty$).

2 Basic Inequalities

- Markov's inequality. Let Z be a *non-negative* random variable. Then $\mathbb{P}[Z \ge t] \le \frac{\mathbb{E}[Z]}{t}$.
  - Proof: Suppose for simplicity that Z is continuous with density f_Z (if Z is discrete, just replace integrals by summations below). Then:

$$\begin{aligned}\mathbb{P}[Z \ge t] &= \int_0^\infty f_Z(z)\mathbf{1}\{z \ge t\}dz \\ &\le \int_0^\infty \frac{z}{t} f_Z(z)\mathbf{1}\{z \ge t\}dz \\ &\le \int_0^\infty \frac{z}{t} f_Z(z)dz \\ &= \frac{\mathbb{E}[Z]}{t}.\end{aligned}$$

  - Note that this result definitely doesn't hold in general for RVs that can take negative values (e.g., take $Z \sim N(0,1)$ as a counter-example).

- Markov's inequality applied to functions: Let ϕ denote any *non-decreasing* and *non-negative* function. Let Z be any random variable. Then Markov's inequality gives

$$\mathbb{P}[Z \ge t] \le \mathbb{P}[\phi(Z) \ge \phi(t)] \le \frac{\mathbb{E}[\phi(Z)]}{\phi(t)},$$

  where the first inequality uses the non-decreasing property, and the second uses Markov's inequality and the non-negative property.

- Chebyshev's inequality: Choose $\phi(t) = t^2$, and replace Z by $|Z - \mathbb{E}[Z]|$. Then

$$\mathbb{P}\big[|Z - \mathbb{E}[Z]| \ge t\big] \le \frac{\mathrm{Var}[Z]}{t^2}.$$

- Chernoff bound: Choose $\phi(t) = e^{\lambda t}$ where $\lambda \geq 0$. Then we have

$$\mathbb{P}[Z \geq t] \leq e^{-\lambda t}\mathbb{E}[e^{\lambda Z}].$$

Despite being a simple application of Markov's inequality, this bound is extremely useful.

3 Simplifying the Chernoff Bound

Rewriting the bound.

- The log-moment-generating function $\psi_Z(\lambda)$ of a random variable Z is defined as

$$\psi_Z(\lambda) = \log \mathbb{E}[e^{\lambda Z}], \quad \lambda \geq 0.$$

Observe that the Chernoff bound above can be written as $\mathbb{P}[Z \geq t] \leq e^{-(\lambda t - \psi_Z(\lambda))}$.

  - Note: If $\mathbb{E}[e^{\lambda Z}] = \infty$ for some λ, then this value of λ does not give a meaningful bound (but a smaller λ might be OK). If Z is sufficiently heavy-tailed, it could even be that $\mathbb{E}[e^{\lambda Z}] = \infty$ for *all* $\lambda > 0$, in which case, the Chernoff bound cannot be used.

- The Cramér transform of Z is defined as

$$\psi_Z^*(t) = \sup_{\lambda \geq 0} \big(\lambda t - \psi_Z(\lambda)\big). \tag{1}$$

By a direct substitution, setting $\lambda = 0$ would make the right-hand term zero, so since we are maximizing over all $\lambda \geq 0$, we conclude that $\psi_Z^*(t) \geq 0$ for all t.

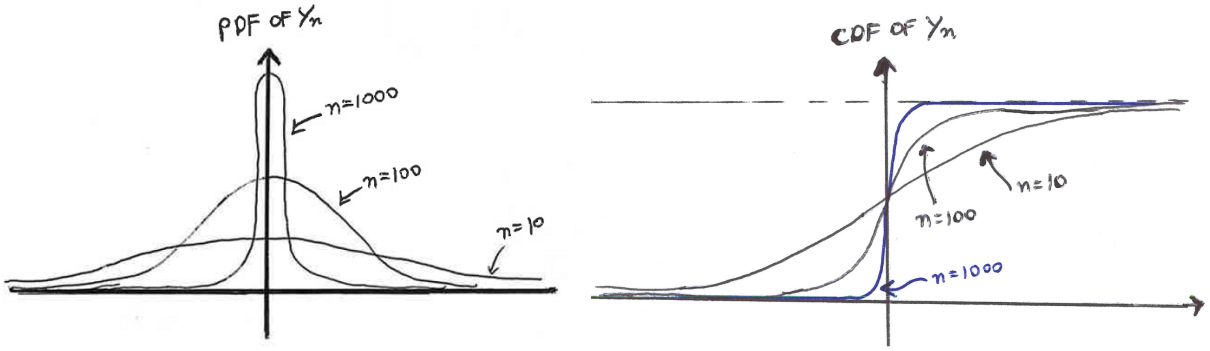
- By simply optimizing over all λ in the Chernoff bound, we have for any random variable Z that

$$\mathbb{P}[Z \geq t] \leq \exp(-\psi_Z^*(t)).$$

This is known as the *Cramér-Chernoff Inequality*.

Sums of independent random variables.

- Let $Z = X_1 + \cdots + X_n$ where $\{X_i\}_{i=1}^n$ are independent and identically distributed (i.i.d.). We expect sharper concentration of $Y_n = \frac{Z}{n}$ as n increases:



- *Chebyshev's inequality on the sum:* We have $\mathrm{Var}[Z] = n\mathrm{Var}[X]$ (by the i.i.d. assumption), and hence Chebyshev's inequality with $t = n\epsilon$ gives

$$\mathbb{P}\left[\frac{1}{n}\big|Z - \mathbb{E}[Z]\big| \geq \epsilon\right] \leq \frac{\mathrm{Var}[X]}{n\epsilon^2}.$$

  - This is an $O\big(\frac{1}{n}\big)$ probability of a "large" deviation, which can be useful but is typically not the best possible.

- *Cramér-Chernoff inequality on the sum:* We have

$$\psi_Z(\lambda) = \log \mathbb{E}[e^{\lambda Z}] = \log \mathbb{E}\left[e^{\lambda \sum_{i=1}^n X_i}\right] = \log \mathbb{E}\left[\prod_{i=1}^n e^{\lambda X_i}\right]$$
$$= \log \prod_{i=1}^n \mathbb{E}\big[e^{\lambda X_i}\big] = \log\Big(\mathbb{E}\big[e^{\lambda X}\big]\Big)^n = n\psi_X(\lambda),$$

  where in the second line we used independence and then the identical distribution property. Then the Cramér-Chernoff inequality with $t = n\epsilon$ gives

$$\mathbb{P}[Z \geq n\epsilon] \leq \exp\big(-n\psi_X^*(\epsilon)\big). \tag{2}$$

  - This is looking better – exponential decay!
  - But $\psi_X^*(\epsilon)$ is a bit complicated (it is not a closed-form formula, and it involves an optimization over λ) – can we simplify further?

- *A simple case: Gaussian random variables.*

  - Let $X \sim \mathcal{N}(0, \sigma^2)$.
  - A direct computation yields $\psi_X(\lambda) = \frac{\lambda^2\sigma^2}{2}$ (this requires a bit of integration).
  - Substituting into (1), we get the expression $\lambda t - \frac{\lambda^2\sigma^2}{2}$. Setting the derivative to zero gives the optimal $\lambda^* = \frac{t}{\sigma^2}$, and hence $\psi_X^*(t) = \frac{t^2}{2\sigma^2}$.
  - Therefore,

$$\mathbb{P}[X \geq t] \leq \exp\left(-\frac{t^2}{2\sigma^2}\right).$$

    Since X and $-X$ have the same distribution, the union bound $\Pr[A \cup B] \leq \Pr[A] + \Pr[B]$ gives

$$\mathbb{P}[|X| \geq t] \leq 2\exp\left(-\frac{t^2}{2\sigma^2}\right).$$

    When we sum n independent copies $Z = X_1 + \cdots + X_n$, analogous reasoning applied to (2) gives

$$\mathbb{P}[|Z| \geq n\epsilon] \leq 2\exp\left(-\frac{n\epsilon^2}{2\sigma^2}\right).$$

    Since this example appears so frequently, it is used as a baseline for a much larger class of distributions with similar concentration.

4 Sub-Gaussian Random Variables and Hoeffding's Inequality

Sub-Gaussian Random Variables.

- From the definition in (1) along with the above Gaussian example, we find that if $\psi_X(\lambda) \leq \frac{\lambda^2\sigma^2}{2}$, then $\psi_X^*(t) \geq \frac{t^2}{2\sigma^2}$. This motivates the following definition.
- **Definition.** A *zero-mean* random variable X is said to be *sub-Gaussian* with parameter σ^2 if $\psi_X(\lambda) \leq \frac{\lambda^2\sigma^2}{2}$, $\forall \lambda > 0$. Denote the set of all such random variables by $\mathcal{G}(\sigma^2)$.
  - Note: Sub-Gaussian variables are very "light-tailed" (tails decaying like e^{-ct^2}). Similar concepts also exists for distributions whose tails are less light, notably including *sub-exponential* (tails decaying like e^{-ct} e.g., see Vershynin's book).
- Properties of sub-Gaussian random variables:
  1. $\mathbb{P}[|X| \geq t] \leq 2\exp\left(-\frac{t^2}{2\sigma^2}\right)$ (as we already proved for Gaussians)
  2. If $X_i \in \mathcal{G}(\sigma_i^2)$ are independent, then $\sum_{i=1}^n a_i X_i \in \mathcal{G}\big(\sum_{i=1}^n a_i^2\sigma_i^2\big)$ (just like with Gaussians)

  The straightforward proofs of these properties are omitted.
  - Combining these properties (with $t = n\epsilon$), we find that if $Z = X_1 + \ldots + X_n$ where the X_i are independent and sub-Gaussian with parameter σ^2, then

$$\mathbb{P}[|Z| \geq n\epsilon] \leq 2\exp\left(-\frac{n\epsilon^2}{2\sigma^2}\right),$$

    just like the sum of n independent Gaussians.
- Equivalent definitions: Sometimes checking whether $\psi_X(\lambda) \leq \frac{\lambda^2\sigma^2}{2}$ can be difficult, e.g., because the MGF is complicated or has no closed-form expression. To verify the sub-Gaussian property, it is useful to note that the following statements are all equivalent for zero-mean X:
  1. (MGF) There exists $K_0 > 0$ such that $\psi_X(\lambda) \leq K_0^2\lambda^2$ for all $\lambda > 0$ (i.e., the above definition of sub-Gaussianity with $K_0^2 = \frac{\sigma^2}{2}$).
  2. (Tail Behavior) There exists $K_1 > 0$ such that $\mathbb{P}[|X| \geq t] \leq 2\exp\left(-\frac{t^2}{K_1^2}\right)$ for all $t \geq 0$.
  3. (Moments) There exists $K_2 > 0$ such that $\mathbb{E}[|X|^p]^{1/p} \leq K_2\sqrt{p}$ for all $p \geq 1$.

  The proofs are omitted here (e.g., see Proposition 2.5.2 of Vershynin's book). The quantities K_0, K_1, K_2 may differ in general, but they all match to within a constant factor (and thus all play a similar role as σ above).

Bounded Random Variables.

- An important class of sub-Gaussian random variables is the class of bounded random variables.
- **Theorem.** Let X be a random variable with $\mathbb{E}[X] = 0$, taking values in a bounded interval $[a, b]$. Then we have $X \in \mathcal{G}\big(\frac{(b-a)^2}{4}\big)$.
  - A proof outline is below, with the details left as an optional appendix.

- Using this result and the first sub-Gaussian property above, we find that for $X \in [a, b]$,

$$\mathbb{P}\big[|X - \mathbb{E}[X]| > t\big] \leq 2\exp\bigg(-\frac{2t^2}{(b-a)^2}\bigg).$$

  - Although the theorem assumed $\mathbb{E}[X] = 0$, we can always replace X by $X - \mu$ and $[a, b]$ by $[a - \mu, b - \mu]$, so the difference between the upper and lower limit is still $b - a$.

  Using a similar argument along with the fact that sums of sub-Gaussian variables are sub-Gaussian, we obtain the following.

- **Corollary (Hoeffding's inequality)** Let $Z = X_1 + \cdots + X_n$, where the X_i are independent and supported on $[a_i, b_i]$. Then

$$\mathbb{P}\bigg[\frac{1}{n}\big|Z - \mathbb{E}[Z]\big| > \epsilon\bigg] \leq 2\exp\bigg(-\frac{2n\epsilon^2}{\frac{1}{n}\sum_{i=1}^{n}(b_i - a_i)^2}\bigg).$$

  When we have all $a_i = a$ and all $b_i = b$, this simplifies to

$$\mathbb{P}\bigg[\frac{1}{n}\big|Z - \mathbb{E}[Z]\big| > \epsilon\bigg] \leq 2\exp\bigg(-\frac{2n\epsilon^2}{(b-a)^2}\bigg),$$

  or even more simply, in the commonly-encountered scenario where $a = 0$ and $b = 1$, we have

$$\mathbb{P}\bigg[\frac{1}{n}\big|Z - \mathbb{E}[Z]\big| > \epsilon\bigg] \leq 2e^{-2n\epsilon^2}.$$

- To keep the expressions simple, we discuss the latter case in further detail. This bound can be viewed as fixing (ϵ, n) and asking how small the deviation probability is. It is also to keep in mind two equivalent statements:

  - If we want the deviation probability to be upper bounded by some $\delta > 0$, and ϵ is given, then we get the following condition by setting $2e^{-2n\epsilon^2} = \delta$ and re-arranging:

$$n \geq \frac{1}{2\epsilon^2}\log\frac{2}{\delta}.$$

    This amounts to fixing (ϵ, δ) and asking how large n needs to be be.

  - Similarly, we can fix (δ, n) and ask what the smallest possible ϵ could be, giving $\epsilon = \sqrt{\frac{1}{2n}\log\frac{2}{\delta}}$. This is consistent with how the Central Limit Theorem shows that most of the probability is within $O\big(\frac{1}{\sqrt{n}}\big)$ of the true mean.

5 (**Optional**) Proof Outline: Bounded RVs are Sub-Gaussian

- Main steps of the proof.

  1. Prove that $\mathrm{Var}[Z] \leq \frac{(b-a)^2}{4}$ for any Z bounded on $[a, b]$.
  2. Show $\psi_X(0) = 0$, $\psi_X'(0) = 0$, and $\psi_X''(\lambda) = \mathrm{Var}[Z]$, where Z is a random variable with PDF $f_Z(z) = e^{-\psi_X(\lambda)}e^{\lambda z}f_X(z)$; hence $\psi_X''(\lambda) \leq \frac{(b-a)^2}{4}$ by Step 1.

3. Taylor expand $\psi_X(\lambda) = \psi_X(0) + \lambda\psi_X'(0) + \frac{\lambda^2}{2}\psi_X''(\theta)$ (for some $\theta \in [0, \lambda]$) and substitute Step 2 to upper bound this by $\frac{\lambda^2}{2} \cdot \frac{(b-a)^2}{4}$.

- The details are given in the appendix of this document.

6 Example Applications

Example 1: Estimating Population Statistics

- Suppose that we have a huge population of voters for an upcoming two-party election, and we want to accurately predict the proportion that will vote for Party A. (Note: We can adapt this example to estimating other things, like prevalence of a disease.)
- Strategy: Choose a voter uniformly at random[3] and ask how they will vote (assuming they will respond honestly). Repeat this n times, and let $\hat{p}$ be the fraction of those n that responded Party A.
- Question: How large should n be to ensure (via Hoeffding's inequality) that $|p^* - \hat{p}| \leq 0.02$ with probability at least 0.95, where p^* is the true proportion and $\hat{p}$ is the estimate?
- Analysis: The n binary observations $X_1, \dots, X_n$ (1 if Party A, 0 if Party B) are Bernoulli(p^*), and letting $Z = X_1, \dots, X_n$, it follows that $\hat{p} = \frac{Z}{n}$ and $p^* = \frac{\mathbb{E}[Z]}{n}$. Hence, we can apply Hoeffding's inequality directly to get

$$\mathbb{P}\big[|\hat{p} - p^*| > 0.02\big] \leq 2e^{-2n(0.02)^2}.$$

  Equating this with 0.05 and re-arranging, we get that $n = \lceil \frac{1}{2(0.02)^2} \log \frac{2}{0.05} \rceil = 4612$ suffices.

- Exercise: Adapt this example to a scenario where every voter answers honestly with probability exactly 0.9 (independently of all other questions/answers).

Example 2: Typical Sequences.

- Let $(U_1, \dots, U_n)$ be i.i.d. random variables drawn from a PMF P_U. Assume that U is integer-valued and finite, only taking values $\{1, \dots, m\}$ for some integer m.
- Question. How many occurrences of each value $u \in \{1, \dots, m\}$ occur?
- Let $Z_u = \sum_{i=1}^n \mathbf{1}\{U_i = u\}$. This is a sum of i.i.d. random variables bounded within $[0, 1]$, and $\mathbb{E}[Z_u] = nP_U(u)$. So by Hoeffding's inequality,

$$\mathbb{P}\big[\big|Z_u - nP_U(u)\big| \geq n\epsilon\big] \leq 2e^{-2n\epsilon^2}.$$

- Since there are m values that U can take, the union bound gives

$$\mathbb{P}\Big[\bigcup_{u=1,\dots,m} \Big\{\big|Z_u - nP_U(u)\big| \geq n\epsilon\Big\}\Big] \leq 2m \cdot e^{-2n\epsilon^2}.$$

  Re-arranging, we find that probability is upper bounded by $\delta > 0$ under the choice $\epsilon = \sqrt{\frac{\log \frac{2m}{\delta}}{2n}}$. Equivalently, if $n \geq \frac{1}{2\epsilon^2} \log \frac{2m}{\delta}$, then the above probability is at most δ.

[3] Probably the main reason that actual polls can be quite inaccurate is that the people they poll are *not* uniformly random.

- The above findings can be viewed in at least two ways:
  - With high probability, all of the counts are within $O(\sqrt{n \log m})$ of their mean as n grows large.
  - For the counts to deviate from their mean by at most $n\epsilon$ with high probability, it suffices to have $n = \text{constant} \times \frac{\log m}{\epsilon^2}$ samples.

Example 3: Graph Degree.

- As an exercise, see if you can use the analysis of Example 1 to bound the maximum degree in a random graph with high probability.
  - More precisely, consider a random graph with n nodes, in which each given edge is present with probability p (independent from all other edges). The edges have no direction, so there are $\binom{n}{2}$ potential edges, and the average number of edges is $p\binom{n}{2}$.
  - The degree of a node is defined as the number of edges attached to that node. For a given node, its mean is $(n-1)p$. The maximum degree of the graph is the highest degree among the n nodes.

Example 4: Randomized Algorithms

- As an exercise, you can try the following: Suppose that a randomized algorithm produces the correct output with probability at least 2/3. Show that by independently running the algorithm n times and letting the final output be the one output the highest number of times, we can boost the success probability to any target $1-\delta$ with a number of trials satisfying $n = O\big(\log \frac{1}{\delta}\big)$.

Example 5: Estimation Under Heavy-Tailed Noise.

- A fundamental primitive in statistics and related areas is estimating the mean of a random variable from independent samples (i.e., given $X_1, \ldots, X_n$ each drawn from P_X, estimate $\mu = \mathbb{E}[X]$).
- If $X - \mu$ is sub-Gaussian for $X \sim P_X$, then we know that the empirical mean $\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} X_i$ works well, with $e^{-cn\epsilon^2}$ decay of the probability of being ϵ-far from the correct value (for some constant c).
- What if we only know that $\mu = \mathbb{E}[X]$ and $\sigma^2 = \text{Var}[X]$ are finite, but the higher moments may be infinite? This occurs for *heavy tailed* distributions, which are often used to model outliers in the data.
- At first glance it looks hopeless to consider Hoeffding's inequality (as X is not only unbounded but heavy-tailed!), but in fact with a more clever choice of estimator, all we need is Hoeffding's and Chebyshev's inequality, and we can get the sub-Gaussian level of accuracy mentioned above!
- The more clever estimator is called *median of means*:
  - Split the n samples into K blocks of size B, so that $n = KB$ (we will ignore rounding/divisibility issues here and below, as they are insignificant for the interesting regimes of parameter choices)
  - For $k = 1, \ldots, K$, let $\hat{\mu}_k$ be the empirical mean computed using only the samples in block k.
  - The final estimate is $\hat{\mu} = \text{median}(\hat{\mu}_1, \ldots, \hat{\mu}_K)$.
- By the definition of median, if $|\hat{\mu} - \mu| > \epsilon$, then at least half of the values in $\hat{\mu}_1, \ldots, \hat{\mu}_K$ must be ϵ-far from μ. Hence,

$$\mathbb{P}\big[|\hat{\mu} - \mu| > \epsilon\big] \le \mathbb{P}\Big[\sum_{k=1}^{K} Z_k \ge \frac{K}{2}\Big],$$

  where Z_k equals 1 if $|\hat{\mu}_k - \mu| > \epsilon$, and $Z_k = 0$ otherwise.

- Defining $p_\epsilon = \mathbb{E}[Z_k] = \mathbb{P}[|\hat{\mu}_k - \mu| > \epsilon]$ and $t = \frac{1}{2} - p_\epsilon$, the above right-hand side is equivalent to

$$\mathbb{P}\Big[\sum_{k=1}^{K}(Z_k - p_\epsilon) \geq Kt\Big],$$

  and as long as $t > 0$ (which we will verify shortly), this is upper bounded by e^{-2Kt^2} by Hoeffding's inequality. Substituting back $t = \frac{1}{2} - p_\epsilon$, we have proved that

$$\mathbb{P}\big[|\hat{\mu} - \mu| > \epsilon\big] \leq e^{-2K(1/2-p_\epsilon)^2}.$$

- Next, by the definition $p_\epsilon = \mathbb{P}[|\hat{\mu}_k - \mu| > \epsilon]$ and the fact that $\hat{\mu}_k$ is the empirical average of B samples, we can simply apply Chebyshev's inequality to obtain $p_\epsilon \leq \frac{\sigma^2}{B\epsilon^2}$, and substituting $B = \frac{n}{K}$ gives $p_\epsilon \leq \frac{K\sigma^2}{n\epsilon^2}$. In particular, if we choose $K = \frac{n\epsilon^2}{4\sigma^2}$, we get $p_\epsilon \leq \frac{1}{4}$ (which gives the desired property $t > 0$ mentioned above), and the previous display equation becomes

$$\mathbb{P}\big[|\hat{\mu} - \mu| > \epsilon\big] \leq e^{-K/8} = \exp\Big(-\frac{n\epsilon^2}{32\sigma^2}\Big).$$

  This is the desired sub-Gaussian style concentration! (Note: The factor of 32 can be improved via a more careful analysis)

- **Caveats/discussion:**
  - Setting $K = \frac{n\epsilon^2}{4\sigma^2}$ requires knowing ϵ and σ in advance, which may be questionable. On the other hand, setting the above upper bound $e^{-K/8}$ to a target value δ gives $K = 8\log\frac{1}{\delta}$, so we can actually set K given only knowledge of such a target δ, which may be more natural.
  - Also note that since K should be an integer, more care is needed with rounding if (e.g.) $\frac{n\epsilon^2}{4\sigma^2} < 1$; the above result may not hold as stated in such scenarios.

Other uses:

- See "Randomized Algorithms" (Motwani/Raghavan) for an entire book on randomized algorithms where these techniques are prominent.
- See https://www.comp.nus.edu.sg/~scarlett/CS5339_notes/09-Theory_Notes.pdf for the use of Hoeffding's inequality in statistical learning theory, a theoretical branch of machine learning.
- (This list could be made much longer!)

Other useful concentration bounds:

- As a simple example of where Hoeffding's inequality can be weaker than ideal, suppose that we are interested in the probability that Binomial(n, p) takes value 0. This event has probability $(1-p)^n \leq e^{-pn}$, Hoeffding's inequality would only give a bound of e^{-2p^2n}, which is much weaker when p is small. The concentration bounds below serve as alternatives that circumvent this weakness.
- Bernstein's inequality: If $Z = \sum_{i=1}^{n}(X_i - \mathbb{E}[X_i])$ with the X_i being i.i.d. and satisfying $|X_i| \leq M$ (with probability one), then

$$\mathbb{P}[Z \geq t] \leq \exp\Big(-\frac{\frac{1}{2}t^2}{\sum_{i=1}^{n}\mathbb{E}[X_i^2] + \frac{1}{3}Mt}\Big).$$

For example, when $X_i \sim \text{Bernoulli}(p)$ we have $M = 1$ and $\mathbb{E}[X_i^2] = p$, and setting $t = \epsilon n$ gives a bound of $\exp\big(-\frac{n\epsilon^2}{2(p+\epsilon/3)}\big)$. This is "Hoeffding-like" when p is "large", but has better $e^{-\Theta(n\epsilon)}$ behavior when p is "small" (e.g., $p \le \epsilon \ll 0$).

  - Note: More general forms of Bernstein's inequality consider *all* moments of the random variable, and drop the requirement of the random variable being bounded. See for example Chapter 2 of Vershynin's textbook.

- Binomial tail bounds: Since the binomial distribution arises especially frequently, it's useful to highlight some of its most widely-used tail bounds. Letting $Z \sim \text{Binomial}(n, p)$ (i.e., Z is the sum of n independent Bernoulli(p) variables), we have the following:

  - The Chernoff bound can be simplified to give

$$\begin{aligned} \mathbb{P}[Z \le \gamma n] &\le e^{-nD(\gamma\|p)} \quad \text{if } \gamma \le p \\ \mathbb{P}[Z \ge \gamma n] &\le e^{-nD(\gamma\|p)} \quad \text{if } \gamma \ge p, \end{aligned}$$

    where $D(a\|b) = a\log\frac{a}{b} + (1-a)\log\frac{1-a}{1-b}$ (see *KL divergence* or *relative entropy* in the upcoming lecture on information theory). These bounds are usually tight to within a $\Theta\big(\frac{1}{\sqrt{n}}\big)$ factor (which is usually insignificant compared to the exponential terms).

  - The following weakened bounds are often more "user-friendly":

$$\begin{aligned} \mathbb{P}[Z \ge (1+\delta)np] &\le \exp\big(-np\big((1+\delta)\log(1+\delta) - \delta\big)\big) \quad \text{for } \delta > 0 \\ \mathbb{P}[Z \le (1-\delta)np] &\le \exp\big(-np\big((1-\delta)\log(1-\delta) + \delta\big)\big) \quad \text{for } \delta \in (0,1) \end{aligned}$$

    These can also be further weakened to the following particularly simple bounds:

$$\begin{aligned} \mathbb{P}[Z \ge (1+\delta)np] &\le \exp\left(-np\cdot\frac{1}{3}\delta^2\right) \quad \text{for } \delta \in (0,1) \\ \mathbb{P}[Z \le (1-\delta)np] &\le \exp\left(-np\cdot\frac{1}{3}\delta^2\right) \quad \text{for } \delta \in (0,1). \end{aligned}$$

    All four of these are particularly useful when p is small (e.g., decreasing as n increases), in which case Hoeffding's inequality may not be powerful enough.

7 Beyond Sums of Independent Random Variables

In many scenarios throughout machine learning and statistics, we would like to establish concentration random variables that are not sums of independent random variables. This is often much more difficult, but there exist tools for this purpose – below are just two examples (without proofs).

Bounded differences and McDiarmid's inequality.

- A function $f : \mathcal{X}^n \to \mathbb{R}$ has the bounded differences property if, for some positive $c_1, .., c_n$,

$$\sup_{x_1,\ldots,x_n,x_i' \in \mathcal{X}} |f(x_1, .., x_i, ..., x_n) - f(x_1, ..., x_i', ..., x_n)| \le c_i$$

for all $i = 1, \ldots, n$. This means that changing any single input value does not change the output value too much.

- Example 1: Let $V = \{1, \cdots, n\}$, and let G be a random graph such that each pair $i, j \in V$ is independently connected with probability p. Let

$$X_{ij} = \begin{cases} 1 & (i,j) \text{ are connected} \\ 0 & \text{otherwise.} \end{cases}$$

The *chromatic number* of G is the minimum number of colors needed to color the vertices such that no two connected vertices have the same color. Writing

$$\text{chromatic number } = f(X_{11}, \cdots, X_{ij}, \cdots, X_{nn}),$$

we find that f satisfies the bounded difference property with $c_{ij} = 1$. This is because adding (resp., removing) an edge at most amounts to needing to add (resp. being able to remove) one color.

- Example 2: Suppose that we throw m balls into n bins uniformly at random. Let $X_1, \ldots, X_m$ be random variables giving the bin indices of the balls. Then if we are interested in a function $f(x_1, \ldots, x_m)$ such as *the number of empty bins* or *the number of bins with at least 2 balls*, we clearly have that $f(\cdot)$ changes by at most one whenever a single index X_i changes. Thus, we again have the bounded differences property with $c_i = 1$.

- **Theorem (McDiarmid's Inequality).** Let $X_1, ..., X_n$ be independent random variables, and let f satisfy the bounded differences property with c_i's. Then

$$\mathbb{P}\big(|f(X_1, ..., X_n) - \mathbb{E}[f(X_1, ..., X_n)]| \geq t\big) \leq 2\exp\bigg(-\frac{2t^2}{\sum_{i=1}^n c_i^2}\bigg).$$

- This is a very useful generalization of Hoeffding's inequality (which is recovered from this result by choosing $f(x_1, \ldots, x_n) = \sum_{i=1}^n x_i$ when the random variables satisfy $X_i \in [a_i, b_i]$ with $c_i = b_i - a_i$).

- **Example (kernel density estimation):** Suppose that we have i.i.d. samples $X_1, \ldots, X_n$ with each X_i being drawn from some probability density function $\psi(x)$, and we would like to know the function ψ (as best we can). The Kernel Density Estimation (KDE) method does this by estimating

$$\widehat{\psi}(x) = \frac{1}{n}\sum_{i=1}^n K(x - X_i)$$

for some "kernel" K such that $\int_{-\infty}^{\infty} K(z)dz = 1$ (e.g., a Gaussian-like curve, so that we try to put more density near each X_i but "smooth it out" away from that point). Suppose that we are interested in the overall error $Z = \int_{-\infty}^{\infty} |\psi(x) - \widehat{\psi}(x)|dx$.

Observe that $Z = f(X_1, \ldots, X_n)$, where the function f is non-linear and quite complicated. Despite this, we can easily deduce its concentration behavior – whenever a single data point is changed, only a single term in $\frac{1}{n}\sum_{i=1}^n K(x - X_i)$ gets affected, and since K integrates to one we can easily check that

$$|f(x_1, \ldots, x_i, \ldots, x_n) - f(x_1, \ldots, x_i', \ldots, x_n)| \leq \frac{2}{n}.$$

Hence, McDiarmid's inequality with $c_i = \frac{2}{n}$ gives the concentration bound

$$\mathbb{P}\big(|Z - \mathbb{E}[Z]| \geq t\big) \leq 2e^{-nt^2},$$

which signifies very sharp concentration around the average when n is large.

- Note: Understanding $\mathbb{E}[Z]$ itself may still be difficult (and strongly dependent on K and ψ), but whatever value it happens to be, we now know that there is sharp concentration around it.

(Optional**) Martingales and Azuma's inequality.**

- A sequence $Y_1, \ldots, Y_n$ is said to be a *martingale* if it holds for each time step i that $\mathbb{E}[Y_{i+1} \,|\, Y_1, \ldots, Y_i] = Y_i$ (no matter which $Y_1, \ldots, Y_i$ values we condition on).[4]
  - In other words, conditioned on the sequence so far, the next element neither increases nor decreases *on average*.
  - Example: If $Y_i = \sum_{i=1}^{n} X_i$ where $X_1, \ldots, X_n$ are independent zero-mean random variables, then it's easy to check that $Y_1, \ldots, Y_n$ is a martingale. (But there are many other examples of martingales for which the increments $Y_{i+1} - Y_i$ are not independent.)
  - Intuition: If a gambler is playing in a *fair* casino, then what happened so far may impact which games they play or how large their bets are, creating some dependence structure. But no matter which (fair) games or what bet sizes, on average there is no gain or loss. (In reality, there would always be an average loss, leading to a related notion called a *super-martingale*.)
    - Naturally, this concept arises for similar reasons in *sequential decision-making* problems in theoretical computer science, machine learning, etc.
  - Doob martingale: Although seemingly different, the martingale idea is closely related to McDiarmid's inequality (in fact, the theorem below can be used to prove McDiarmid's inequality). Briefly, this connection stems from the *Doob martingale*, where for a random variable of the form $A = f(Z_1, \ldots, Z_n)$ (with independent Z_i's) we define $X_i = \mathbb{E}[A|Z_1, \ldots, Z_i]$. This can easily be shown to produce a martingale. As a concrete example, if $Z_1, \ldots, Z_n$ were edges in a random graph, then the sequence $X_1, \ldots, X_n$ would have an interpretation of *revealing the edges one-by-one* and seeing how some property $f(\cdot)$ develops (e.g., number of triangles formed in the graph).
- **Theorem (Azuma's Inequality).** If $Y_0, Y_1, \ldots, Y_n$ is a martingale and it holds with probability one that $|Y_{i+1} - Y_i| \leq c_i$ for all i (i.e., *bounded increments*), then it holds that

$$\mathbb{P}\big[|Y_n - Y_0| \geq n\epsilon\big] \leq 2\exp\left(-\frac{2n\epsilon^2}{\frac{1}{n}\sum_{i=1}^{n} c_i^2}\right).$$

  - The extra term Y_0 is included for convenience, and whether or not it's included is just a matter of renaming (e.g., shifting n to $n+1$). We could also just specialize to the case that $Y_0 = 0$.
  - In the case of independent sums $Y_i = Y_0 + \sum_{i=1}^{n} X_i$, if we assume that $X_i \in [a_i, b_i]$, then we get $c_i = b_i - a_i$, and the result precisely reduces to Hoeffding's inequality.

[4] Strictly speaking it is also required that $\mathbb{E}[|Y_i|]$ is finite for each i, but this is a minor technical condition.

(**Optional**) Appendix: Proving Bounded RVs are Sub-Gaussian

Claim: Any bounded random variable $Z \in [a, b]$ has variance at most $\text{Var}[Z] \leq \frac{(b-a)^2}{4}$.
Proof:

- It suffices to show $\text{Var}[Z] \leq \frac{1}{4}$ when $Z \in [0, 1]$, since then the general case follows by shifting and re-scaling.
- We have

$$\mathbb{E}[(Z - c)^2] = \mathbb{E}[Z^2] - 2c\mathbb{E}[Z] + c^2,$$

  which is minimized at $c = \mathbb{E}[Z]$ (check by setting the derivative to zero). Therefore, $\text{Var}[Z] \leq \mathbb{E}[(Z-c)^2]$ for any c.
- Setting $c = \frac{1}{2}$ and using the fact that $Z \in [0, 1]$, we conclude that $\text{Var}[Z] \leq \frac{1}{4}$, as required.

Claim: (Recall that the log moment generating function is defined as $\psi_X(\lambda) = \log \mathbb{E}[e^{\lambda X}],\ \lambda \geq 0$.) Assuming that $\mathbb{E}[X] = 0$, we have $\psi_X(0) = 0$, $\psi'_X(0) = 0$, and $\psi''_X(\lambda) = \text{Var}[Z]$ for any $\lambda > 0$, where Z is a random variable with PDF $f_Z(z) = e^{-\psi_X(\lambda)} e^{\lambda z} f_X(z)$. (Note: Z implicitly depends on λ)
Proof:

- (i) Since $\psi_X(\lambda) = \log \mathbb{E}[e^{\lambda X}]$, we have $\psi_X(0) = \log 1 = 0$.
- (ii) By direct differentiation, we have $\psi'_X(\lambda) = \frac{\mathbb{E}[Xe^{\lambda X}]}{\mathbb{E}[e^{\lambda X}]}$, which implies $\psi'_X(0) = \mathbb{E}[X] = 0$ (recall that we assumed $\mathbb{E}[X] = 0$ above).
- (iii) Differentiating a second time, we have $\psi''_X(\lambda) = \frac{\mathbb{E}[X^2 e^{\lambda X}] \cdot \mathbb{E}[e^{\lambda X}] - \mathbb{E}[Xe^{\lambda X}]^2}{\mathbb{E}[e^{\lambda X}]^2}$. To simplify this, note that for any function $g(x)$, we have

$$\begin{aligned}
\mathbb{E}[g(X)e^{\lambda X}] &= \int f_X(x) e^{\lambda x} g(x)\, dx \\
&= \int f_Z(x) e^{-\psi_X(\lambda)} g(x)\, dx \\
&= e^{-\psi_X(\lambda)} \int f_Z(x) g(x)\, dx \\
&= \mathbb{E}[e^{\lambda X}]\mathbb{E}[g(Z)],
\end{aligned}$$

  where we applied the definition of Z, factored out the constant $e^{-\psi_X(\lambda)}$, and substituted the definition of ψ_X. It follows that $\psi''_X(\lambda) = \mathbb{E}[Z^2] - \mathbb{E}[Z]^2$, which is simply $\text{Var}[Z]$.

Claim: $\psi_X(\lambda) \leq \frac{\lambda^2}{2} \cdot \frac{(b-a)^2}{4}$ for any zero-mean random variable taking values in $[a, b]$. In other words, X is sub-Gaussian with parameter $\frac{(b-a)^2}{4}$.
Proof:

- By a (particular form of the) second-order Taylor expansion, we have

$$\psi_X(\lambda) = \psi_X(0) + \lambda \psi'_X(0) + \frac{\lambda^2}{2} \psi''_X(\theta)$$

  for some $\theta \in [0, \lambda]$.

- The claim is then immediate from the previous two claims upon noticing that by the definition of $f_Z(z) = e^{-\psi_X(\lambda)} e^{\lambda z} f_X(z)$, the random variable Z inherits X's property of taking values on $[a, b]$. (Outside that range, f_X is zero and hence so is f_Z.)

CS5275 Lecture 3: The Probabilistic Method

Jonathan Scarlett

January 26, 2025

Acknowledgment. The first version of these notes was prepared by Eugene Lim and Ivona Martinović for a CS6235 assignment.

Useful References

- Chapter 6 of Mitzenmacher's and Upfal's book *Probability and Computing*
- Class 11 and 12 of Wooter's lecture *Randomized Algorithms*[1]
- Postle's lecture series *Probabilistic Methods*[2]
- Alon's and Spencer's book *The Probabilistic Method*
- Matoušek's and Vondrák's lecture notes *The Probabilistic Method*[3]
- Luke Postle's video lectures[4]

Categorization of material:

- Core material: Sections 1–5 except Distinct Sum example (but going over that is encouraged) and parts labeled as optional. You can also skip the $\lfloor 2^{k/2} \rfloor$ proof in Theorem 2.1.
- Extra material: Distinct Sum, Lovasz Local Lemma, Appendix

(Exam will strongly focus on "Core". Take-home assessments may occasionally require consulting "Extra".)

1 Introduction

The probabilistic method is a technique for proving the existence of certain objects (e.g., graphs, codes, algorithms) having certain properties, using probabilistic arguments (even when the object itself may have nothing to do with probability). The idea is to define a probability space and subsequently demonstrate that the probability that a randomly selected object has the desired properties is greater than zero. This, in turn, implies that the probability space must encompass such an object, thereby confirming the existence of such an object. More broadly, similar ideas of probabilistic reasoning can also be used to prove other kinds of results such as non-existence properties and bounds.

We will see several examples throughout the lecture, but mention a couple as motivation:

[1] https://www.youtube.com/playlist?list=PLkvhuSoxwjI_JL7GYcJHK7-EK55t0KYGO
[2] https://www.youtube.com/playlist?list=PL2BdWtDKMS6nRF72s3TOGyBqXwMVHYiLU
[3] https://www.cs.cmu.edu/~15850/handouts/matousek-vondrak-prob-ln.pdf
[4] https://www.youtube.com/playlist?list=PL2BdWtDKMS6nRF72s3TOGyBqXwMVHYiLU

- A fundamental problem in Coding Theory (to be covered later) is to find a "large" collection of strings (say, binary of length n) that are "well-separated" (say, differing pairwise in at least δn positions for some $\delta \in (0,1)$). A concrete example would be: *Find a set $S \subseteq \{0,1\}^n$ of size $|S| = 2^{0.1n}$ such that any two of them differ in at least $0.1n$ positions.*
  - Note: If you are unfamiliar with coding theory, think of this as a company wanting to produce many serial numbers ($2^{0.1n}$ of them) under the requirement that no two of them can be too similar.

  A simple argument based on Hoeffding's inequality and the union bound shows that by generating $2^{0.1n}$ completely random strings, this "well-separated" property will hold with high probability when n is large enough. Thus, at least one such set S must exist.

- In areas such as Machine Learning, it is of interest to map "high-dimensional data" (say $x_1, \ldots, x_N$ with $x_i \in \mathbb{R}^d$ for large d) to some lower-dimensional space (say $\mathbb{R}^m$ with $m \ll d$) while preserving some key structural properties of the original data. One important structural property is pairwise distances, and the resulting problem is: *Find a matrix $A \in \mathbb{R}^{m \times d}$ such that*

$$(1-\epsilon)\|x_i - x_j\| \leq \|Ax_i - Ax_j\| \leq (1+\epsilon)\|x_i - x_j\|, \quad \forall i, j.$$

  Finding such an A by hand is very tricky, but it turns out that by letting A have i.i.d. Gaussian entries, this property will hold with high probability under scaling of the form $m = O\big(\frac{1}{\epsilon^2}\log N\big)$. You can look up the *Johnson–Lindenstrauss Lemma* if you are interested to know more.

2 Basic Counting Arguments

The basic counting argument is an integral part of the probabilistic method. The idea of the counting method is as follows:

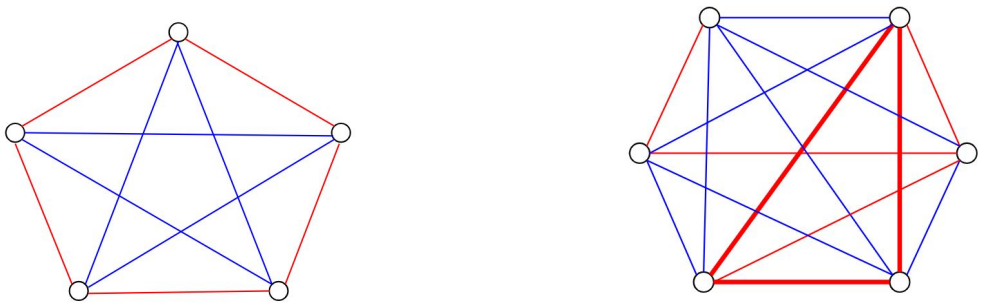
1. Set up a sampling scheme to sample an element (e.g., a random draw of S or A in the above examples) that has the potential to satisfy the desired property.
2. Design a set of "bad events" such that if none of the events hold, the sampled object x definitely has the desired property.
3. Count the number of such bad events. Let this number be m.
4. Compute the probability that each of these bad events occurs (or an upper bound on this probability). Let this probability be p.
5. Conclude that the object of interest must exist if $1 - mp > 0$. This is because, by union bound, we know the probability that none of the bad events holds is at least $1 - mp$.

2.1 Ramsey Number

Section 1.1 of *The Probabilistic Method*

The Ramsey number $R(k,l)$ is the smallest positive integer n such that, in every two-coloring (say blue and red) of the edges of a complete graph K_n of n vertices, we can always find either a red clique K_k of size k or a blue clique K_l of size l.

- Note: Another interpretation is to replace "red edge" by "edge is present" and "blue edge" by "'edge is absent", and to seek either a clique (i.e., fully connected subset) of size k or an independent set (i.e., subset with no connections) of size l. We'll stick with the red/blue terminology.
- It's fair to say that this is a fairly abstract concept without an immediately obvious application, but Ramsey numbers have had implications throughout theoretical computer science and beyond (e.g., see https://www.cs.umd.edu/~gasarch/TOPICS/ramsey/ramsey.html)



(a) K_5 with no monochromatic K_3.

(b) K_6 with at least one monochromatic K_3.

Figure 1: Example showing $R(3,3) = 6$.

Let us consider a small example where $k = l = 3$. Figure 1a provides a coloring of K_5 with no monochromatic clique of size 3, implying that $R(3,3) > 5$. It turns out that $R(3,3) = 6$. To see this, note that each vertex v in K_6 is adjacent to five edges. By the pigeonhole principle, at least three such edges have the same color. Without loss of generality, we assume these edges are colored blue and are connected to vertices x, y, z. If any of the edges $(x,y),(y,z),(z,x)$ are colored blue, then we have a blue clique of size 3; otherwise, these edges are all colored red, thus forming a red clique of size 3. Figure 1b demonstrates an edge-coloring of K_6 with a red clique of size 3.

The Ramsey theorem shows that $R(k,l)$ is always finite, but calculating exact Ramsey numbers for larger values of k and l can be challenging and not many precise values of $R(k,l)$ are known. Instead, we can settle on knowing upper and lower bounds on $R(k,l)$ can be helpful. Here, we will focus on deriving lower bounds for the "diagonal" ($l = k$) Ramsey numbers $R(k,k)$.

Theorem 2.1 (Erdős, 1947)**.** *If* $\binom{n}{k} \cdot 2^{1-\binom{k}{2}} < 1$ *then* $R(k,k) > n$. *Thus for any* $k \geq 3$, $R(k,k) > \lfloor 2^{k/2} \rfloor$.

(Note: A similar upper bound dependent on $4^k = 2^{2k}$ *is known, so while the bounds don't exactly match, they are both exponential and they mainly only differ in the exponent being* $k/2$ *vs.* $2k$*.)*

Proof. Let us consider a random two-coloring of the edges of a complete graph K_n where every edge is colored independently either red or blue with probability $1/2$. Consider an arbitrary ordering of all $\binom{n}{k}$ cliques $\{C_1, \ldots, C_{\binom{n}{k}}\}$ of size k. For each $i \in \{1, \ldots, \binom{n}{k}\}$, let A_i be the event that the clique C_i is monochromatic. For A_i to hold, the $\binom{k}{2}$ edges should be all blue or all red (holding with probability $2^{-\binom{k}{2}}$ each), and hence

$$\Pr[A_i] = 2^{1-\binom{k}{2}}.$$

As there are $\binom{n}{k}$ cliques, we can apply the union bound to calculate the probability that at least 1 such event

occurs:

$$\Pr\left[\bigcup_{i=1}^{\binom{n}{k}} A_i\right] \leq \sum_{i}^{\binom{n}{k}} \Pr[A_i] = \binom{n}{k} \cdot 2^{1-\binom{k}{2}} < 1,$$

where the last inequality holds by assumption in the theorem. Therefore, the probability that no such event A_i occurs is $1 - \binom{n}{k}2^{1-\binom{k}{2}} > 0$; or in other words, there is a positive probability of sampling a two-coloring of K_n without a monochromatic K_k. Thus, $R(k,k) > n$.

If $k \geq 3$ and we take $n = \lfloor 2^{k/2} \rfloor$ then:

$$\binom{n}{k} \cdot 2^{1-\binom{k}{2}} \overset{(i)}{<} \frac{n^k}{k!} \cdot \frac{2^{1+k/2}}{2^{k^2/2}} \overset{(ii)}{\leq} \frac{2^{k^2/2}}{k!} \cdot \frac{2^{1+k/2}}{2^{k^2/2}} = \frac{2^{1+k/2}}{k!} \overset{(iii)}{<} \frac{2^{1+k/2}}{2^k} = 2^{1-k/2} < 1, \tag{1}$$

where step (i) applies $\binom{n}{k < \frac{n^k}{k!}}$ and $\binom{k}{2} = \frac{k(k-1)}{2} = \frac{k^2-k}{2}$, step (ii) applies $n = \lfloor 2^{k/2} \rfloor \leq 2^{k/2}$, and step (iii) applies $k! \geq 2^k$. From (1), we deduce that $R(k,k) > \lfloor 2^{k/2} \rfloor$ as desired. □

Converting into a randomized algorithm. The proof of existence hints at a strategy to sample a two-coloring of a complete graph with n vertices such that there is no monochromatic clique of size k. More generally, many proofs by the probabilistic method can be converted to an algorithm that uses random sampling to try to find an element with the desired property. Such algorithms can be categorized into *Monte Carlo algorithms* or *Las Vegas algorithms*, as we describe below.

Monte Carlo algorithm. A Monte Carlo algorithm is a randomized algorithm that may have some probability of failure, while often having a fixed (pre-specified) number of "rounds" of random sampling. In the simplest case, we simply run a single round of sampling and show that it succeeds with high probability. More generally, we might run the same random procedure multiple times and either (i) figure out which one is the "best" one, or (ii) aggregate all of the results. (For instance, if the randomized algorithm only "succeeds" with probability 0.01, we could run it $\gg 100$ times to be confident of at least one success.)

In the proof, we've already established the sampling algorithm as a means of coloring the graph by independently assigning each edge as either red or blue, selected at random. If the probability of obtaining a sample with the desired property is represented as p. If p is already close to one (say $p = 1 - o(1)$), then simply running the random procedure once will succeed wit high probability.

For example, if we want to color K_{1000} in a way that there are no monochromatic K_{20}, we can first check if this is possible using Theorem 2.1. As $n = 1000 \leq 2^{10} = 2^{k/2}$, this means that there exists a coloring with no monochromatic K_{20}. If we simplify the first expression from the theorem, we can get:

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

where the three inequalities follow from (i) $\binom{n}{k} \leq \frac{n^k}{k!}$ and $\binom{k}{2} = \frac{k(k-1)}{2}$; (ii) the inequality $n \leq 2^{k/2}$ established above; and (iii) $k! > 2^k > 2^{k/2+1}$ for any $k > 2$. Using this simplification, we know that the probability that a random coloring has a monochromatic K_{20} is maximally $\frac{2^{20/2+1}}{20!} < 8.5 \cdot 10^{-16}$. Thus, the suggested Monte Carlo algorithm has only a very small probability of giving an incorrect solution.

Some important caveats to this sort of argument are as follows:

- In situations where the probability of bad events is high, Monte Carlo methods may have a higher probability of providing incorrect solutions and performing poorly.

- Perhaps more importantly, *checking the desired property cannot always be done efficiently.* In fact, the Ramsey number example fits in this category, as checking with a clique of a certain size exists in a graph is a famously (NP-)hard problem. Another example would be the first motivating example in Section 1 – with $|S| = 2^{0.1n}$ strings, the number of pairwise checks would be $O\big(\big(2^{0.1n}\big)^2\big) = O\big(2^{0.2n}\big)$, which is exponentially large. In fact, even just storing the list of strings would be a problem (unless n is small), as there are $2^{0.1n}$ of them.

Las Vegas algorithm. Unlike a Monte Carlo algorithm, a Las Vegas is *guaranteed* to output the correct the solution; the idea is to repeatedly perform the random sampling and continue until it is verified to succeed. Thus, the guaranteed correctness may come at the expense of (having some small probability of) taking a long time, if p is small. More precisely, the number of trials until the first success follows a Geometric(p) distribution, whose average is $1/p$.

In the Ramsey number example, when the value k is constant, there exists a polynomial time verification algorithm: Just search over all $\binom{n}{k}$ cliques to ensure that they are not monochromatic. However, it is important to note that if k grows with n, this verification algorithm ceases to execute within polynomial time. In fact, clique funding is an NP-hard problem.

Weaknesses. While the basic counting method is often relatively straightforward, its efficacy depends on the design of sampling schemes and the accurate identification of bad events. The use of the union bound may also result in overly loose inequalities, especially in cases with a large number of bad events m, where condition $1 - mp > 0$ becomes unlikely to hold. We will see another method that can get around this problem in Section 6.

3 Expectation Arguments

The expectation argument relies on the fact that a discrete random variable must, with positive probability, take on values both lower and higher than its expected value. For instance, if the average score of the class is 42 points, then at least one student scored is 42 or higher, and at least one student scored 42 or lower. This observation forms the basis of the expectation argument in probabilistic reasoning. Formally, we have the following lemma.

Lemma 3.1. *For any real-valued random variable X, we have* $\Pr[X \geq \mathbb{E}[X]] > 0$ *and* $\Pr[X \leq \mathbb{E}[X]] > 0$.

Proof. We present the proof for the case that X is a discrete random variable, but the argument for continuous variables is similar with sums replaced by integrals.

We apply a proof by contradiction: If we were to have $\Pr[X \geq \mathbb{E}[X]] = 0$, it would follow that

$$\mathbb{E}[X] = \sum_x x \Pr[X = x] = \sum_{x < \mathbb{E}[X]} x \Pr[X = x] < \sum_{x < \mathbb{E}[X]} \mathbb{E}[X] \Pr[X = x] = \mathbb{E}[X],$$

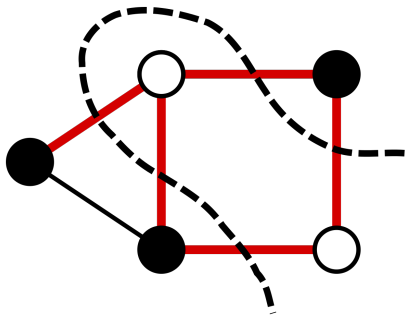
which is a contradiction. The other claim is proven similarly. □

3.1 Finding a Large Cut

Section 6.2.1 of *Probability and Computing*

A cut (S_1, S_2) for an undirected graph $G = (V, E)$ is a partition of V into two disjoint sets S_1 and S_2. The value of the cut $C(S_1, S_2)$ is the number of edges that cross from S_1 to S_2. The following figure[5] shows an example of a large cut where S_1 is the set of black vertices and S_2 is the set of white vertices. The value of the cut $C(S_1, S_2) = 5$, which is also the maximum cut for the graph. More generally, finding the cut with the largest number of edges is known as the Maximum Cut (MAXCUT) problem.

- Note: Applications of MAXCUT include network design, statistical physics, and clustering problems.
- It is an NP-hard problem, but a 0.878-approximation algorithm is known. We'll be less ambitious here and show that a very simple probabilistic argument gives a 0.5-approximation.



Theorem 3.2. *Given an undirected graph $G = (V, E)$ with $|E| = m$, there exist a cut (S_1, S_2) such that the value of the cut $C(S_1, S_2) \geq m/2$.*

Proof. For each vertex $v \in V$, assign v uniformly to either S_1 or S_2. Let $e_1, \ldots, e_m$ be an arbitrary arrangement of E and define $X_i = \mathbf{1}[e_i \text{ connects } S_1 \text{ to } S_2]$ for each $i \in [m]$ where $[m] = \{1, \ldots, m\}$. Since the probability that e_i crosses S_1 to S_2 is 0.5, we have $\mathbb{E}[X_i] = 0.5$. Then

$$\mathbb{E}[C(S_1, S_2)] = \mathbb{E}\left[\sum_{i=1}^{m} X_i\right] = \sum_{i=1}^{m} \mathbb{E}[X_i] = \frac{m}{2}.$$

Since $\mathbb{E}[C(S_1, S_2)] = m/2$, by Lemma 3.1, there exist a cut (S_1, S_2) such that $C(S_1, S_2) \geq m/2$. □

Tightness of the lower bound. It turns out that $m/2$ is an essentially tight lower bound for the size of the maximum cut. To see this, consider the complete graph K_{2n} with $2n$ vertices where $n \in \mathbb{N}$, and let (S_1, S_2) be a cut with $|S_1| = k$ and $|S_2| = 2n - k$. Note that the value of this cut is $k(2n - k)$, which is maximized when $k = n$. As such, the value of any maximum cut in K_{2n} is n^2. Since $m = \binom{2n}{2} = n(2n - 1) = 2n^2 - n$, some simple algebraic manipulation reveals that the value of the maximum cut is $m/(2 - 1/n)$, which approaches $m/2$ as $n \to \infty$.

[5] Source: Wikipedia page on maximum cut.

Converting to a randomized algorithm. As before, we can convert this argument into a Las Vegas algorithm that finds a cut (S_1, S_2) with $C(S_1, S_2) \geq m/2$ in expected polynomial time: we just have to sample many random cuts $(S_1^{(1)}, S_2^{(1)}), (S_1^{(2)}, S_2^{(2)}), \ldots$ until we obtain an $(S_1^{(T)}, S_2^{(T)})$ with $C(S_1^{(T)}, S_2^{(T)}) \geq m/2$. Since checking if a given $S_1^{(j)}, S_2^{(j)}$ satisfy $C(S_1^{(j)}, S_2^{(j)}) \geq m/2$ takes polynomial time, we are left to show that $\mathbb{E}[T]$ is also polynomial in m. Let $p = \Pr[C(S_1, S_2) \geq m/2]$. Observe that

$$\frac{m}{2} = \mathbb{E}[C(S_1, S_2)] = \sum_{i<m/2} i \Pr[C(S_1, S_2) = i] + \sum_{i \geq m/2} i \Pr[C(S_1, S_2) = i] \leq (1-p)\left(\frac{m}{2} - 1\right) + pm,$$

where the inequality follows since $i \leq \frac{m}{2} - 1$ in the first sum and $i \leq m$ in the second sum. By re-arranging the above inequality $\frac{m}{2} \leq (1-p)\left(\frac{m}{2} - 1\right) + pm$ and solving for p, we obtain $p \geq \frac{1}{m/2+1}$. If we imagine each sampling of the cut as a (biased) coin flip, then we are interested in the following question: "Given a coin that flips a head with probability p, how many flips T do we need in expectation before we get a head?" Since T follows the geometric distribution with parameter p, we have $\mathbb{E}[T] = 1/p \leq m/2 + 1$, which is linear in m. Note also that this is only an upper bound, and the actual value of $\mathbb{E}[T]$ may be much smaller.

Converting to a deterministic algorithms. This turns out to be a well-known example where the randomized algorithm can be *derandomized*, i.e., a deterministic strategy gives the same guarantee. See Appendix A for the details. Deterministic algorithms are often considered preferable, e.g., so that the practical performance can be more predictable or reliable. However, in general performing derandomization while maintaining computational efficiency can be difficult or impossible.

3.2 Maximum Satisfiability

Section 6.2.2 of *Probability and Computing*

Here we generalize the example given in the introduction section of this lecture.

In a logical formula, a literal is either a variable x or its complement $\bar{x}$, and a clause is a disjunction ($\vee$) of literals. A logical formula is in conjunctive normal form (CNF) if it is the conjunction ($\wedge$) of a set of clauses. For brevity, we shall call a formula that is in CNF form a *CNF formula*. For example, the CNF formula

$$\psi_0(x_1, x_2, x_3, x_4) = (x_1 \vee x_2 \vee \bar{x}_3 \vee \bar{x}_4) \wedge (\bar{x}_1 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee x_4)$$

has four variables and three clauses.

Let ψ be a CNF formula with n variables. A truth assignment to ψ is an assignment of its variables $x_1, \ldots, x_n$ to values $\{\texttt{TRUE}, \texttt{FALSE}\}$ such that $\psi(x_1, \ldots, x_n) = \texttt{TRUE}$. For example, a truth assignment to ψ_0 as defined above is $x_1 = x_3 = \texttt{FALSE}$ and $x_2 = x_4 = \texttt{TRUE}$. The problem of finding a truth assignment to ψ is known as the Constraint Satisfaction (SAT) problem. Furthermore, if all clauses have exactly k literals, then the problem is known as k-SAT. We will discuss more about k-SAT in a later section.

For now, let us consider a variation of the problem where we want to find an assignment to the variables that satisfy as many clauses as possible. This is known as the Maximum Satisfiability (MAXSAT) problem. (Both SAT and MAXSAT are famously NP-hard problems, and SAT is the basis for countless reductions showing other problems to be NP-hard.)

Theorem 3.3. *Let ψ be a CNF formula with m clauses, and let k_i be the number of literals in the i-th*

clause. Let $k = \min_{i\in[m]} k_i$. Then there is an assignment whose number of satisfied clauses is at least

$$\sum_{i=1}^{m}(1-2^{-k_i}) \geq m(1-2^{-k}).$$

Proof. Assign each variable independently and uniformly to either `TRUE` or `FALSE`. Let Y_i be the indicator random variable for which the i-th clause is satisfied. We have $\mathbb{E}[Y_i] = 1-2^{-k_i}$, and thus

$$\mathbb{E}\left[\sum_{i=1}^{m} Y_i\right] = \sum_{i=1}^{m}\mathbb{E}[Y_i] = \sum_{i=1}^{m}(1-2^{-k_i}) \geq m(1-2^{-k}).$$

By Lemma 3.1, there must exist an assignment with at least that many clauses satisfied. □

Converting to a randomized algorithm. Similarly to the MAXCUT example, we can convert such an argument into a Las Vegas algorithm that finds an assignment with at least $m(1-2^{-k})$ satisfied clauses. The conversion scheme and proof are similar to before. The reader is encouraged to verify that we need to sample, on average, at most $m \cdot 2^{-k} + 1$ assignments to obtain the desired assignment.

Weaknesses. The expectation method relies on the expectation of the quantity of interest to be sufficiently large (or sufficiently small, depending on the specifics of the problem). However, in some cases, extreme outliers in the constructed probability space might skew this expectation to be way smaller (or larger) than what is required. In such cases, one can sometimes condition on suitably-chosen events that hold with high probability but 'remove" the outlier cases. The use of second moments in addition to the mean is also often useful; see Section 4.

4 Second Moment Methods

Just like expectation, the variance (a.k.a., second centered moment) is another useful statistic for a random variable X. In the context of the probabilistic method, *second moment methods* refer to methods that use both mean and variance information to prove existence properties. Perhaps the most common tool for this purpose of Chebyshev's inequality, which is stated as follows.

Lemma 4.1. *(Chebyshev's inequality). Let X be a random variable with a finite variance. Then for $t > 0$*

$$\Pr[|X-\mathbb{E}[X]| \geq t] \leq \frac{\mathrm{Var}[X]}{t^2}.$$

Proof. Observe that

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

where the inequality holds due to Markov's inequality. □

Notes: Computing or upper bounding $\mathrm{Var}[X]$ is not always straightforward; it is worth noting the following:

- If $X = \sum_{i=1}^{n} U_i$ and the U_i are independent, we simply have $\mathrm{Var}[X] = \sum_{i=1}^{n}\mathrm{Var}[U_i]$ (an easy case).

- If $X = \sum_{i=1}^{n} U_i$ but the U_i are not necessarily independent, then this generalizes to: (try as an exercise)

$$\mathrm{Var}[X] = \sum_{i=1}^{n} \mathrm{Var}[U_i] + \sum_{(i,j)\,:\,i \neq j} \mathrm{Cov}[U_i, U_j],$$

  and bounding the second tern may be more difficult (but often possible).

- Alternatively, if $X = \sum_{i=1}^{n} U_i$ then we can decompose $\mathrm{Var}[X] = \mathbb{E}[X^2] - \mathbb{E}[X]^2$ and use $\mathbb{E}[X^2] = \sum_{i=1}^{n} \sum_{j=1}^{n} \mathbb{E}[U_i U_j]$ (try as an exercise); see the tutorial for an example ('Counting Triangles').

- There are also more advanced tools for bounding variance (e.g., Efron-Stein inequality), but they are beyond our scope.

4.1 Bounding the Middle Binomial Coefficient

Section 5.2 of *Matoušek's and Vondrák's Lecture Notes*

The binomial coefficient $\binom{2m}{m}$ is the largest among the binomial coefficients $\binom{2m}{k}$ for $k = 0, 1, \ldots, 2m$. It frequently appears in various mathematical formulas, so having access to simple bounds can be very useful. A very simple upper bound is $\binom{2m}{m} \leq 2^{2m}$, which follows by interpreting $\binom{2m}{m}$ as the number of length-$2m$ sequences containing exactly m ones and m zeros (and the upper bound 2^{2m} is the *total* number of binary sequences). Using the second moment method, we can show that this simple upper bound is tight to within a $\Theta(\sqrt{m})$ factor.

Theorem 4.2. *For all $m \geq 1$, we have $\binom{2m}{m} \geq 2^{2m}/(4\sqrt{m}+2)$.*

Proof. Let us define a random variable $X = X_1 + X_2 + \cdots + X_{2m}$, where we independently sample each X_i as 0 or 1 with probability 1/2. That is, $X \sim \mathrm{Binomial}(2m, 1/2)$We know that the expected value of each X_i is 1/2. Using the linearity of expectation, we can calculate the expected value of X as $\mathbb{E}[X] = m$. Similarly, we can calculate variance for each variable $\mathrm{Var}[X_i] = \mathbb{E}[X_i^2] - \mathbb{E}[X_i]^2 = \frac{1}{2} - (\frac{1}{2})^2 = \frac{1}{4}$. Since $X_1, \ldots, X_{2m}$ are independent, the variance of X is $\mathrm{Var}[X] = (1/4) \cdot 2m = m/2$. Now, if we apply Chebyshev's inequality with $t = \sqrt{m}$, we get

$$\Pr[|X - m| < \sqrt{m}] \geq \frac{1}{2}.$$

We rewrite the probability as

$$\Pr[|X - m| < \sqrt{m}] = \sum_{x\,:\,|x-m|<\sqrt{m}} \Pr[X = x] = \sum_{k\,:\,|k|<\sqrt{m}} \Pr[X = m + k]$$

where in the last step, we replace $x - m$ by k. Since $X \sim \mathrm{Binomial}(2m, 1/2)$, we know that the probability of X being equal to a specific value $m + k$ is $\binom{2m}{m+k} \cdot 2^{-2m} \leq \binom{2m}{m} \cdot 2^{-2m}$ (because $\binom{2m}{m}$ is the largest binomial coefficient). Therefore,

$$\frac{1}{2} \leq \sum_{k\,:\,|k|<\sqrt{m}} \Pr[X = m + k] \leq (2\sqrt{m}+1)\binom{2m}{m} 2^{-2m},$$

where the $2\sqrt{m} + 1$ term comes from upper bounding how many integers k satisfy $|k| < \sqrt{m}$. Simple re-arranging gives the desired bound, $\binom{2m}{m} \geq 2^{2m}/(4\sqrt{m}+2)$. □

4.2 Distinct Sum Problem

Section 2.1 of https://cse.buffalo.edu/~hungngo/classes/2011/Spring-694/lectures/sm.pdf

Consider the following problem: *Given an integer n, what's the largest-cardinality subset $S \subseteq \{1, \dots, n\}$ we can find such that every $S' \subseteq S$ has a different value of $\sum_{i \in S'} i$?* Let $f(n)$ denote this largest possible cardinality, and we say that the set S satisfies the *distinct sum property*.

It is easy to see that

$$f(n) \geq \lfloor \log_2 n \rfloor + 1$$

because we can choose powers of two $S = \{1, 2, 4, 8, \dots\}$. Any distinct combination of powers two will give a distinct value; this follows readily from the fact that the integers can be converted to binary in a one-to-one manner, and we can interpret $S' \subseteq S$ as indicating which bits are set to 1.

Using the probabilistic method, a useful upper bound on $f(n)$ can be proved. Note that unlike previous sections where we gave existence statements, this result is a *non-existence* statement, stating that no suitable sets S above a certain size can exist.

Theorem 4.3. *It holds that $f(n) \leq \log_2 n + \frac{1}{2} \log_2 \log_2 n + O(1)$.*

Proof. For intuition, we first prove a weaker result that has $\log_2 \log_2 n$ in place of $\frac{1}{2} \log_2 \log_2 n$. Let S be a set satisfying the distinct sum property, and let k be its cardinality. Since $S \subseteq \{1, \dots, n\}$, the sums can only take values below nk. But by definition there must be 2^k distinct sums, so it must hold that $2^k \leq nk$. Performing some simple asymptotic manipulations on this inequality gives $k \leq \log_2 n + \log \log n + O(1)$ (see the above link if you want details on this step).

We refine this argument using Chebyshev's inequality. This time we explicitly write $S = \{a_1, \dots, a_k\}$. We let S' be a *random* subset in which each element of S is included with probability $\frac{1}{2}$, and define the sum $X = \sum_{i \in S'} i$. Observe that for any integer i, there are only two possibilities:

- If i cannot be produced by summing any subset in S, then $\Pr[X = i] = 0$.
- If i can be produced by summing some subset in S, then $\Pr[X = i] = \frac{1}{2^k}$. This is because the 2^k subsets are equally likely and each give a distinct sum.

Letting $\mu = \mathbb{E}[X]$ and $\sigma^2 = \mathrm{Var}[X]$, Chebyshev's inequality gives

$$\Pr\left[|X - \mu| \leq 2\sigma\right] \geq 1 - \frac{1}{4} = \frac{3}{4}.$$

On the other hand, using the above property $\Pr[X = i] \in \left\{0, \frac{1}{2^k}\right\}$ and the fact that there are at most $4\sigma + 1$ integers satisfying $|X - \mu| \leq 2\sigma$, we have

$$\Pr\left[|X - \mu| \leq 2\sigma\right] \leq \frac{4\sigma + 1}{2^k}.$$

Combining the above inequalities gives $\frac{4\sigma+1}{2^k} \geq \frac{3}{4}$.

Furthermore, since X is an independent sum of variables of the form $a_i \times \text{Bernoulli}(1/2)$ (for $i = 1, \dots, k$), and since the variance of Bernoulli(1/2) is 1/4, we get

$$\sigma^2 = \frac{\sum_{i=1}^k a_i^2}{4} \leq \frac{n^2 k}{4},$$

which implies $\sigma \leq n\sqrt{k}/2$. (Note that the last step above follows since $a_i \leq n$.)

Combining the conclusions of the above two paragraphs gives $\frac{2n\sqrt{k}+1}{2^k} \geq \frac{3}{4}$, or equivalently

$$n \geq \frac{(3/4)2^k - 1}{2\sqrt{k}}.$$

From here, some asymptotic manipulations similar to those of the simpler argument (first paragraph of this proof) give $k \leq \log_2 n + \frac{1}{2}\log\log n + O(1)$, as desired. □

4.3 (**Optional**) Other Applications and Inequalities

To highlight how diverse applications of the probabilistic method can be, we briefly mention an application to number theory. Informally, the result proved is that *except for a vanishingly small number of positive integers n, it holds that the number of unique prime factors is close to* $\log\log n$. See `https://www.youtube.com/watch?v=5pV_35vjVmU` for a precise statement and proof.

The second moment is also often used to derive threshold properties of random graphs; specifically, if each edge is independently included in the graph with probability p, then the graph satisfies a property of interest with probability approaching 1 for p above a suitable threshold, but with probability approach 0 for p below a similar threshold. See the tutorial for an example on the existence or non-existence of triangles depending on whether $p \gg \frac{1}{n}$ or $p \ll \frac{1}{n}$.

We also briefly mention that Chebyshev's inequality is not the only second moment based tool. To name one other example, the Paley-Zygmund Inequality (Notes 6 of *Measure-Theoretic Probability*) states that for any random variable $X \geq 0$ and constant $0 \leq \theta \leq 1$, it holds that

$$\Pr[X > \theta\mathbb{E}[X]] \geq (1-\theta)^2 \frac{\mathbb{E}[X]^2}{\mathbb{E}[X^2]}.$$

This is related to variance since $\mathrm{Var}[X] = \mathbb{E}[X^2] - \mathbb{E}[X]^2$, so $\frac{\mathbb{E}[X]^2}{\mathbb{E}[X^2]}$ being high (e.g., close to 1) indicates lower variance. Specializing to $\theta = 0$ gives $\Pr[X > 0] \geq \frac{\mathbb{E}[X]^2}{\mathbb{E}[X^2]}$, which is particularly useful when we only need to show that a certain object exists (i.e., $X > 0$, where X counts occurrences of that object).

5 Sample and Modify Approaches

Sometimes building a random structure with the desired properties directly can be complicated. In such cases, a simpler solution may be to construct a random structure that lacks the desired properties and subsequently modifying it so that it has those properties. An illustration of this approach, known as the sample-and-modify technique, is given below.

5.1 Independent Sets

Section 6.4.1 of *Probability and Computing*

As a reminder, a subset S of vertices in a graph G is called an independent set if, for any pair of vertices in S, there does not exist an edge between them. Finding independent sets in graphs is of both theoretical interest, with applications including network design, data clustering, and resource allocation (e.g., an edge may indicate that two objects are "conflicting", and we want to find non-conflicting subsets).

Identifying the largest independent set within a graph is a computationally challenging problem. However, the following theorem illustrates how the probabilistic method can offer valuable bounds and insights into

the size of the largest independent set in a graph.

Theorem 5.1. *Let $G = (V, E)$ be a connected graph on n vertices with $m \geq n/2$ edges. Then G has an independent set with at least $n^2/4m$ vertices.*

Proof. Let us define the average degree of the vertices as $d = 2m/n \geq 1$. We can define the following sample-and-modify algorithm:

1. **Sample.** Select a subset S of vertices from G that we "tentatively" intend to keep; specifically, each of the n vertices is independently placed in S with probability $1/d$. Keep only the edges that connect these selected vertices.

2. **Modify.** For each remaining edge (between two nodes in S), delete it, and also delete one of its neighboring vertices. The final independent set is the set of non-deleted vertices in S.

As we have deleted all of the edges, the remaining vertices form an independent set, which we constructed by first sampling the vertices and then modifying the remaining graph.

The analysis is as follows:

- Let X be the number of selected vertices in the first step. Given that the graph comprises n vertices, and considering that each vertex is selected with a probability $1/d$, it follows that $\mathbb{E}[X] = \frac{n}{d}$.

- Let Y be the number of edges that remain after the first step. Since there are $m = nd/2$ edges in the graph, and an edge remains if and only if its two adjacent vertices are selected, it follows that $\mathbb{E}[Y] = \frac{nd}{2}(\frac{1}{d})^2 = \frac{n}{2d}$.

- In the second step, we delete all of the edges and in the worst case, for each remaining edge we delete a different vertex, for a maximum total of Y vertices. At the end of the algorithm, the size of an independent set that we construct is at least $X - Y$, and thus $\mathbb{E}[X - Y] = \frac{n}{d} - \frac{n}{2d} = \frac{n}{2d}$. By the expectation argument, there exists a scenario in which $X - Y \geq \frac{n}{2d}$.

We have thus constructed a random independent set with at least $n/2d$ vertices on average, so by the expectation argument, there must exist an independent set of size at least $n/2d = n^2/4m$. □

6 Lovász Local Lemma

This section builds on the counting argument described in Section 2, which was based on the following general strategy to argue that entities with some property $\mathcal{P}$ exist:

1. Design a sampling scheme $\mathcal{S}$ and let $s \sim \mathcal{S}$ be an arbitrary sample.

2. Design a set of bad events $\mathcal{B}_1, \ldots, \mathcal{B}_m$ for which if none of these bad events holds, then s satisfy $\mathcal{P}$. Below we will let $\bar{\mathcal{B}}_i$ denote the complement of $\mathcal{B}_i$.

3. Show that there is a positive probability that none of the bad events occur.

In the third step, we need to find a lower bound for the probability that none of the bad events occur, which is equivalent to upper bounding the probability that least one of them occurs. In particular, if there is some $p \in [0, 1]$ such that $\Pr[\mathcal{B}_i] \leq p$ for all $i \in \{1, \ldots, m\}$, then by the union bound, we have $\Pr[\cap_{i=1}^m \bar{\mathcal{B}}_i] \geq 1 - mp$. If $mp \leq 1$, then we are done – this is essentially the approach used to prove Theorem 2.1.

The Lovász local lemma is a useful result to facilitate the final step of the argument if it is not possible to achieve $mp \leq 1$. To motivate the lemma, suppose (unrealistically but only momentarily) that the bad events are mutually independent. Then, it is easy to show that there is a positive probability that none of the bad events occur: We simply have $\Pr[\cap_{i=1}^{m} \bar{\mathcal{B}}_i] \geq (1-p)^m$, which is positive if $p \neq 1$. Unfortunately, this argument is almost never suitable, as the bad events are almost always dependent.

Intuitively, the Lovász local lemma states that if $\mathcal{B}_1, \ldots, \mathcal{B}_m$ are "not too dependent", then we can treat them as "roughly independent". The vague notions of "not too dependent" and "roughly independent" are made formal in the following definition and theorem.

Definition 6.1. Let $S = \{\mathcal{B}_1, \ldots, \mathcal{B}_m\}$. For each $i \in [m]$, we say that an event $\mathcal{B}_i$ is mutually independent to all but at most d events if there exist a set $S_{\text{dep}} \subset S \backslash \{\mathcal{B}_i\}$ (possibly different for each i) of size at most d such that for any subset $S_{\text{ind}} \subset S \backslash (S_{\text{dep}} \cup \{\mathcal{B}_i\})$, the probability of event $\mathcal{B}_i$ is unchanged upon conditioning on the (complements of the) events indexed by S_{ind}:

$$\Pr\left[\mathcal{B}_i \middle| \bigcap\nolimits_{\mathcal{B} \in S_{\text{ind}}} \bar{\mathcal{B}}_i\right] = \Pr[\mathcal{B}_i].$$

(Recall that $\bar{\mathcal{B}}_i$ denotes the complement event of $\mathcal{B}_i$.)

Theorem 6.2 (Lovász local lemma)**.** *Let $\mathcal{B}_1, \ldots, \mathcal{B}_m$ be a set of bad events such that for each $i \in [m]$, we have $\Pr[\mathcal{B}_i] \leq p$ and $\mathcal{B}_i$ is mutually independent to all but at most d events. If $4pd \leq 1$, then*

$$\Pr\left[\bigcap_{i=1}^{m} \bar{\mathcal{B}}_i\right] \geq (1-2p)^m > 0.$$

Intuitively, the condition $4pd \leq 1$ states that the more dependence is permitted (higher d), the rarer the events have to be (smaller p) in order to maintain the "independence-like" property. To see the need for this kind of a condition, suppose that we were to have $pd = 1$. Then, a possible scenario would be that in which there are d events with probability $\frac{1}{d}$ each that are *disjoint* from each other (an extreme form of dependence), meaning exactly one of them always holds. Then we would have $\Pr\left[\bigcup_{i=1}^{m} \mathcal{B}_i\right] = 1$ or equivalently $\Pr\left[\bigcap_{i=1}^{m} \bar{\mathcal{B}}_i\right] = 0$, which contradicts our goal of showing that this probability is positive.

The (optional) proof of Theorem 6.2 is given in Appendix B. We proceed to look at an example application.

6.1 k-Satisfiability

Section 6.7.2 of *Probability and Computing*

We now return to constraint satisfaction problem introduced at the start of the lecture (as well as Section 3.2), in which there are n variables $x_1, \ldots, x_n$, and m clauses $c_1, \ldots, c_m$, with each clause being the OR of a subset of variables and/or negations of variables. Previously we discussed the problem of satisfying as many clauses as possible (the MAXSAT problem), whereas here we ask the question of whether or not it is possible to satisfy *all* clauses (the SAT problem).

Accordingly, let $\psi = c_1 \wedge \ldots \wedge c_n$, which is called a *conjunctive normal form* (CNF) formula. We call ψ a k-CNF formula if all of its clauses have exactly k literals (i.e., variables or their negations). The problem of determining whether or not a satisfying assignment to a k-CNF formula exists is known as k-SAT. The following theorem uses the probabilistic method to give a general sufficient condition under which such an assignment is guaranteed to exist.

Theorem 6.3. *If none of the variables in a k-CNF formula ψ appear in more than $2^k/4k$ clauses, then there exists a satisfying assignment for ψ.*

- Note: Remarkably, the number of variables n and clauses m play no direct role! (Though the condition in the theorem does prevent m from growing arbitrarily large.)
- This result is vacuous for $k \leq 4$ (since any such k gives $\frac{2^k}{4k} \leq 1$), but becomes stronger as k increases.

Proof. Assign each variable independently and uniformly to either `TRUE` or `FALSE`. Let $\mathcal{B}_i$ be the bad event that the i-th clause is not satisfied. Thus, $p = \Pr[\mathcal{B}_i] = 2^{-k}$. Observe that $\mathcal{B}_i$ is mutually independent to all other events related to clauses that do not share variables with the i-th clause. Since each variable can appear in at most $2^k/4k$ clauses, $\mathcal{B}_i$ is mutually independent to all but $d \leq k2^k/4k = 2^{k-2}$ events. Since

$$4dp \leq 4 \times 2^{k-2} \times 2^{-k} = 1,$$

by Theorem 6.2, the probability of sampling an assignment with none of the bad events occurring is nonzero. □

Appendix

A Deterministic Algorithm for Finding a Large Cut

Section 6.3 of *Probability and Computing*

We have previously proved the existence of a cut (S_1, S_2) with value $C(S_1, S_2) \geq m/2$ by showing that a randomized strategy gives $\mathbb{E}[C(S_1, S_2)] \geq m/2$. We shall now see how derandomization can help us deterministically obtain a cut that always achieves $C(S_1, S_1) \geq m/2$.

Let $v_1, v_2, \ldots, v_n$ be an arbitrary order for the vertices. We will also designate ϵ_i as the set in which we place v_i, where ϵ_i can be either S_1 or S_2. We have already shown that if ϵ_i are independent with $\Pr(\epsilon_1 = S_1) = \Pr(\epsilon_1 = S_2) = \frac{1}{2}$, then $\mathbb{E}[C(S_1, S_2)] \geq m/2$. To "derandomize" this result, we will consider incrementally choosing each ϵ_i in a deterministic manner based on $\epsilon_1, \ldots, \epsilon_{i-1}$.

Let $\mathbb{E}[C(S_1, S_2)|\epsilon_1, \ldots, \epsilon_k]$ denote the average cut value given specific choices of $\epsilon_1, \ldots, \epsilon_k$ but with the remaining values $\epsilon_{k+1}, \ldots, \epsilon_n$ still being random. We seek to sequentially choose the ϵ_i values in manner that ensures the following:

$$\mathbb{E}[C(S_1, S_2)|\epsilon_1, \epsilon_2, \ldots, \epsilon_{k-1}] \leq \mathbb{E}[C(S_1, S_2)|\epsilon_1, \epsilon_2, \ldots, \epsilon_{k-1}, \epsilon_k]. \tag{2}$$

Before describing how to achieve this goal, we discuss its implications.

Recursively applying the inequality, we find that $\mathbb{E}[C(S_1, S_2)] \leq \mathbb{E}[C(S_1, S_2)|\epsilon_1, \epsilon_2, \ldots, \epsilon_n]$, where $\mathbb{E}[C(S_1, S_2)]$ represents the "purely random" scenario in which no vertices have been assigned to any set, and

$$\mathbb{E}[C(S_1, S_2)|\epsilon_1, \epsilon_2, \ldots, \epsilon_n]$$

represents the "final cut value" after having placed every vertex into either S_1 or S_2. (Hence, the $\mathbb{E}[\cdot|\cdot]$ operation is not actually "averaging" anything – by this point, $C(S_1, S_2)$ is fully specified.) Since $\mathbb{E}[C(S_1, S_2)] \geq m/2$, this implies that our deterministic algorithm will yield a cut with a value of at least $m/2$.

We establish the property (2) through induction. In the base case, when ϵ_1 is introduced, we have $\mathbb{E}[C(S_1, S_2)|\epsilon_1] = \mathbb{E}[C(S_1, S_2)]$. This base case holds by symmetry – it makes no difference whether we place the first node in S_1 or S_2, since the remaining ϵ_i all follow a 50/50 choice anyway. For the inductive step, fix the iteration index k and suppose that $\epsilon_1, \ldots, \epsilon_{k-1}$ have already been specified. At this stage, there are two options for placing v_k: either in S_1 or in S_2, each with a probability of $1/2$. Consequently, we have

$$\mathbb{E}[C(S_1, S_2)|\epsilon_1, \epsilon_2, \ldots, \epsilon_{k-1}] = \frac{1}{2}\mathbb{E}\Big[C(S_1, S_2) \,\Big|\, (\epsilon_1, \epsilon_2, \ldots, \epsilon_{k-1}), \epsilon_k = S_1\Big] + \frac{1}{2}\mathbb{E}\Big[C(S_1, S_2) \,\Big|\, (\epsilon_1, \epsilon_2, \ldots, \epsilon_{k-1}), \epsilon_k = S_2\Big].$$

Since the maximum of two terms is at least as high as the average, it follows that

$$\begin{aligned}\max(\mathbb{E}[C(S_1, S_2)|\epsilon_1, \epsilon_2, \ldots, \epsilon_{k-1}, \epsilon_k = S_1], \mathbb{E}[C(S_1, S_2)|\epsilon_1, \epsilon_2, \ldots, \epsilon_{k-1}, \epsilon_k = S_2]) \\ \geq \mathbb{E}[C(S_1, S_2)|\epsilon_1, \epsilon_2, \ldots, \epsilon_{k-1}].\end{aligned}$$

Therefore, we know that we just need to determine the expectation with which set S_1 or S_2 is greater and put v_k in that set, and (2) will hold.

At this stage, it may not be obvious how to compute the two expectations in the $\max(\cdot, \cdot)$; this is detailed as follows. When calculating $\mathbb{E}[C(S_1, S_2)|\epsilon_1, \epsilon_2, \ldots, \epsilon_{k-1}, \epsilon_k = S_1]$, we already know from the k assignments of ϵ_i that certain edges contribute to the cut, and for the remaining ones, we know that the probability that

its two endpoints will end up in different sets (thus incrementing the cut value by 1) is 1/2 Therefore, to calculate the expectation, using the linearity property, we can just sum the edges that already contribute and half of the remaining ones. This can be computed in linear time, and we can do the same for the other expectation.

Simplifying further, it is straightforward to show that choosing $\epsilon_k = S_1$ or $\epsilon_k = S_2$ is equivalent to placing the vertex v_k in the set where v_k has fewer neighbors. The edges that don't involve v_k contribute equally to both expectations. Therefore, by allocating v_k to the set with fewer neighbors, a greater number of its edges will contribute to the overall cut. With this, we can conclude the formulation of our deterministic algorithm:

- Begin by initially assigning the first vertex arbitrarily to either set S_1 or S_2.
- Subsequently, allocate each successive vertex to the set containing fewer neighbors. (If there is a tie, then either one can be chosen arbitrarily.)

We have demonstrated that this algorithm always gives a cut with minimally $m/2$ edges.

B (**Optional**) Proof and Extensions of the Lovász Local Lemma

Class 11.3 of *Randomized Algorithms*

Proof of Theorem 6.2:
We begin with the following useful lemma.

Lemma B.1. *Let $\mathcal{B}_1, \ldots, \mathcal{B}_m$ be a set of bad events such that for all $i \in [m]$, we have $\Pr[\mathcal{B}_i] \leq p$ and $\mathcal{B}_i$ is mutually independent to all but at most d other events where $4dp \leq 1$. For any set $S \subset \{\mathcal{B}_1, \ldots, \mathcal{B}_m\}$ and any $\mathcal{B}_i \notin S$,*

$$\Pr\left[\mathcal{B}_i \middle| \bigcap_{\mathcal{B} \in S} \bar{\mathcal{B}}\right] \leq 2p.$$

Proof. We prove this by induction. Let $S \subset \{\mathcal{B}_1, \ldots, \mathcal{B}_m\}$ and $\mathcal{B}_i \notin S$. When $|S| = 0$, we have

$$\Pr\left[\mathcal{B}_i \middle| \bigcap_{\mathcal{B} \in S} \bar{\mathcal{B}}\right] = \Pr[\mathcal{B}_i] \leq p \leq 2p$$

where the first inequality holds by assumption in the lemma.

Now suppose the statement holds for all S' with $|S'| \leq k$ and choose S with $|S| = k+1$. Partition S into two sets S_{ind} and S_{dep} such that $\mathcal{B}_i$ is mutually independent to S_{ind} and dependent to S_{dep}. Note that the choice of S_{ind} and S_{dep} might differ for different $\mathcal{B}_i$. If $|S_{\text{ind}}| = k+1$, then $S = S_{\text{ind}}$, which implies that

$$\Pr\left[\mathcal{B}_i \middle| \bigcap_{\mathcal{B} \in S} \bar{\mathcal{B}}\right] = \Pr\left[\mathcal{B}_i \middle| \bigcap_{\mathcal{B} \in S_{\text{ind}}} \bar{\mathcal{B}}\right] = \Pr[\mathcal{B}_i] \leq p \leq 2p.$$

Now suppose $|S_{\text{ind}}| \leq k$. Then using the definition of conditional probability, we have

$$\Pr\left[\mathcal{B}_i \middle| \bigcap_{\mathcal{B} \in S} \bar{\mathcal{B}}\right] = \Pr\left[\mathcal{B}_i \middle| \left(\bigcap_{\mathcal{B} \in S_{\text{dep}}} \bar{\mathcal{B}}\right) \cap \left(\bigcap_{\mathcal{B} \in S_{\text{ind}}} \bar{\mathcal{B}}\right)\right] = \frac{\Pr\left[\mathcal{B}_i \cap \left(\bigcap_{\mathcal{B} \in S_{\text{dep}}} \bar{\mathcal{B}}\right) \middle| \left(\bigcap_{\mathcal{B} \in S_{\text{ind}}} \bar{\mathcal{B}}\right)\right]}{\Pr\left[\left(\bigcap_{\mathcal{B} \in S_{\text{dep}}} \bar{\mathcal{B}}\right) \middle| \left(\bigcap_{\mathcal{B} \in S_{\text{ind}}} \bar{\mathcal{B}}\right)\right]}.$$

The numerator term can be loosely bounded by

$$\Pr\left[\mathcal{B}_i \cap \left(\bigcap_{\mathcal{B} \in S_{\text{dep}}} \bar{\mathcal{B}}\right) \middle| \left(\bigcap_{\mathcal{B} \in S_{\text{ind}}} \bar{\mathcal{B}}\right)\right] \leq \Pr\left[\mathcal{B}_i \middle| \left(\bigcap_{\mathcal{B} \in S_{\text{ind}}} \bar{\mathcal{B}}\right)\right] = \Pr[\mathcal{B}_i] \leq p.$$

The denominator term can be bounded by first observing that the probability of its negation is

$$\begin{aligned}\Pr\left[\left(\bigcup_{\mathcal{B}\in S_{\text{dep}}}\mathcal{B}\right)\middle|\left(\bigcap_{\mathcal{B}\in S_{\text{ind}}}\bar{\mathcal{B}}\right)\right] &\overset{(a)}{\leq} \sum_{\mathcal{B}\in S_{\text{dep}}}\Pr\left[\mathcal{B}\middle|\left(\bigcap_{\mathcal{B}\in S_{\text{ind}}}\bar{\mathcal{B}}\right)\right]\\ &\overset{(b)}{\leq} 2p\cdot|S_{\text{dep}}|\\ &\overset{(c)}{\leq} 2pd\\ &\overset{(d)}{\leq} \frac{1}{2},\end{aligned}$$

where (a) holds due to the union bound, (b) holds due to the inductive hypothesis, (c) holds due to the assumption that $\mathcal{B}_i$ is mutually independent to all but at most d other events, and (d) holds due to the assumption that $4dp \leq 1$. Thus, the denominator can be bounded by

$$\Pr\left[\left(\bigcap_{\mathcal{B}\in S_{\text{dep}}}\bar{\mathcal{B}}\right)\middle|\left(\bigcap_{\mathcal{B}\in S_{\text{ind}}}\bar{\mathcal{B}}\right)\right] = 1-\Pr\left[\left(\bigcup_{\mathcal{B}\in S_{\text{dep}}}\mathcal{B}\right)\middle|\left(\bigcap_{\mathcal{B}\in S_{\text{ind}}}\bar{\mathcal{B}}\right)\right] \geq \frac{1}{2}.$$

Putting them together, we have

$$\Pr\left[\mathcal{B}_i\middle|\bigcap_{\mathcal{B}\in S}\bar{\mathcal{B}}\right] = \frac{\Pr\left[\mathcal{B}_i\cap\left(\bigcap_{\mathcal{B}\in S_{\text{dep}}}\bar{\mathcal{B}}\right)\middle|\left(\bigcap_{\mathcal{B}\in S_{\text{ind}}}\bar{\mathcal{B}}\right)\right]}{\Pr\left[\left(\bigcap_{\mathcal{B}\in S_{\text{dep}}}\bar{\mathcal{B}}\right)\middle|\left(\bigcap_{\mathcal{B}\in S_{\text{ind}}}\bar{\mathcal{B}}\right)\right]} \leq \frac{p}{1/2} = 2p. \qquad \square$$

We now prove Theorem 6.2 using Lemma B.1.

Proof of Theorem 6.2. By Lemma B.1, we have $\Pr[\bar{\mathcal{B}}_i|\bar{\mathcal{B}}_{i+1}\cap\cdots\cap\bar{\mathcal{B}}_m] \geq 1-2p$ for all $i\in[m]$. Thus,

$$\begin{aligned}\Pr\left[\bigcap_{i=1}^{m}\bar{\mathcal{B}}_i\right] &= \Pr\left[\bar{\mathcal{B}}_1\middle|\bigcap_{i=2}^{m}\bar{\mathcal{B}}_i\right]\times\Pr\left[\bar{\mathcal{B}}_2\middle|\bigcap_{i=3}^{m}\bar{\mathcal{B}}_i\right]\times\cdots\times\Pr[\bar{\mathcal{B}}_{m-1}|\bar{\mathcal{B}}_m]\times\Pr[\bar{\mathcal{B}}_m]\\ &\geq (1-2p)^m\\ &> 0,\end{aligned} \qquad \square$$

where the last step holds by assumption in the theorem.

Variations and Extensions:

Other variants of the Lovász local lemma also exist, one of which replaces $4pd\leq 1$ by $ep(d+1)\leq 1$, and more importantly, an *asymmetric version* in which the dependencies are encoded via a general graph in which different nodes (random variables) may have significantly different degrees (number of dependencies). We will not cover such variants.

Another natural follow-up question is whether it is possible to transform an existence argument relying on the Lovász local lemma into an efficient algorithm. In other words, we want a general algorithmic approach that allows us to avoid all the bad events $\mathcal{B}_1,\ldots,\mathcal{B}_m$. While we won't delve into this topic here, we direct interested readers to explore the relevant resources on *algorithmic Lovász Local Lemma* for more information (e.g., Class 12.1 of *Randomized Algorithms* linked on the first page).

CS5275 Lecture 4: Convexity

Jonathan Scarlett

February 18, 2025

Useful references:

- Blog posts on Lagrange multipliers,[1] duality for linear programming,[2] and general Lagrange duality[3]
- Boyd and Vandenberghe's "Convex Optimization" book[4]
- Boyd's lectures on convex optimization, available on YouTube
- Nesterov's lecture notes on convex optimization

Categorization of material:

- Core material: Sections 1–4 except parts involving the Hessian, maxflow-mincut duality, and support vector machine.
- Extra material: Section 5 (KKT conditions) and the above-mentioned parts from Sections 1–4.

(Exam will strongly focus on "Core". Take-home assessments may occasionally require consulting "Extra".)

1 Convex Sets and Functions

Basic definitions.

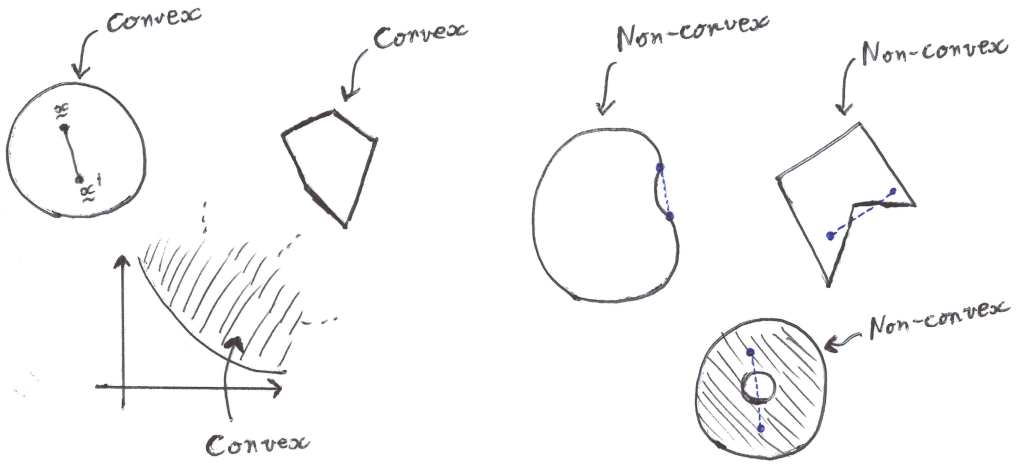
- A set D (e.g., a subset of $\mathbb{R}^d$) is said to be a *convex set* if, for all $\mathbf{x} \in D$ and $\mathbf{x}' \in D$, it holds that

$$\lambda \mathbf{x} + (1-\lambda)\mathbf{x}' \in D$$

  for all $\lambda \in [0,1]$

  - In words (roughly): Draw a straight line between any two points in D. This whole line segment must also lie within D.
  - Examples:

[1] http://jeremykun.com/2013/11/30/lagrangians-for-the-amnesiac/
[2] http://jeremykun.com/2014/06/02/linear-programming-and-the-most-affordable-healthy-diet-part-1/
[3] http://blogs.princeton.edu/imabandit/2013/02/21/orf523-lagrangian-duality/
[4] http://web.stanford.edu/~boyd/cvxbook/

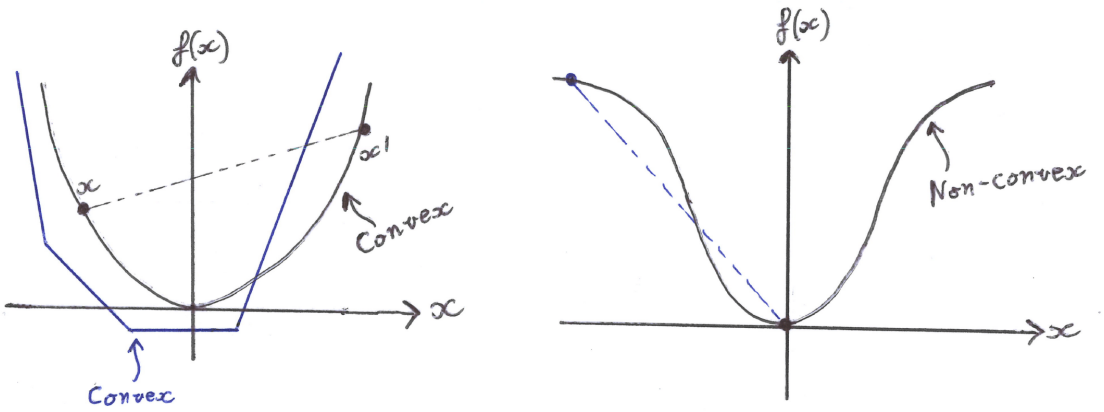


- A function $f : D \to \mathbb{R}$ is said to be a *convex function* if, for all $\mathbf{x} \in D$ and $\mathbf{x}' \in D$, it holds that

$$f(\lambda \mathbf{x} + (1-\lambda)\mathbf{x}') \leq \lambda f(\mathbf{x}) + (1-\lambda) f(\mathbf{x}')$$

  for all $\lambda \in [0, 1]$. Implicitly, this requires that the domain D is a convex set.

  - In words (roughly): Draw a straight line between $(\mathbf{x}, f(\mathbf{x}'))$ and $(\mathbf{x}', f(\mathbf{x}'))$. For inputs in between $\mathbf{x}$ and $\mathbf{x}'$, the function lies below this straight line.
  - Illustration:



  - We say that $f(\mathbf{x})$ is a *concave function* if $-f(\mathbf{x})$ is a convex function.
  - Convex = "bowl-shaped" ($\cup$), concave = "arch-shaped" ($\cap$)
  - A function is simultaneously convex and concave $\iff$ it is affine (i.e., a "straight line" (or plane)).
  - Key property. For a convex function, any local minimum is also a global minimum.

Other examples.

- Convex functions: $\|\mathbf{x}\|^2$, e^x, e^{-x}, $\log \sum_{i=1}^d e^{x_i}$, and many more.
- Concave functions: $-\|\mathbf{x}\|^2$, $\log x$, $\log \det \mathbf{X}$, $\sum_{i=1}^d x_i \log \frac{1}{x_i}$, and many more.

Equivalent definitions of convexity.

- Recall the notions of gradient and Hessian for $\mathbf{x} = [x_1, \ldots, x_d]^T$:

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_d} \end{bmatrix}, \qquad \nabla^2 f = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_d} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_d} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_d \partial x_1} & \frac{\partial^2 f}{\partial x_d x_2} & \cdots & \frac{\partial^2 f}{\partial x_d^2} \end{bmatrix}.$$

- (First order) If f is differentiable, then it is convex if and only if

$$f(\mathbf{x}') \geq f(\mathbf{x}) + \nabla f(\mathbf{x})^T (\mathbf{x}' - \mathbf{x})$$

  for all $\mathbf{x}, \mathbf{x}'$. (The function lies above its tangent plane)

- (Second order) If f is twice differentiable, then it is convex if and only if

$$\nabla^2 f(\mathbf{x}) \succeq \mathbf{0}$$

  for all $\mathbf{x} \in D$. (The Hessian is positive semi-definite)

Operations that preserve convexity.

- If $f_1(\mathbf{x})$ and $f_2(\mathbf{x})$ are convex, and α_1 and α_2 are positive, then $f(\mathbf{x}) = \alpha_1 f_1(\mathbf{x}) + \alpha_1 f_2(\mathbf{x})$ is convex. By induction, a similar statement holds for $\sum_{\ell=1}^{L} \alpha_\ell f_\ell(\mathbf{x})$ also for $L > 2$.

- If $f_1(\mathbf{x}), \ldots, f_L(\mathbf{x})$ are convex, then so is $f(\mathbf{x}) = \max_{\ell=1,\ldots,L} f_\ell(\mathbf{x})$.

- Certain compositions of the form $f(\mathbf{x}) = g(h(\mathbf{x}))$ are convex under certain conditions on g and h (see Section 3.2 of Boyd and Vandenberghe's book)

  - Simplest case: If h is a linear (or affine) function and g is convex, then f is convex.

Jensen's inequality.

- *Jensen's inequality* states that, for any random vector $\mathbf{X}$ and convex function f, it holds that

$$f(\mathbb{E}[\mathbf{X}]) \leq \mathbb{E}[f(\mathbf{X})].$$

  This is used in countless proofs in machine learning, statistics, information theory, etc.

- Note that the inequality is true directly from the definition of convexity when $\mathbf{X}$ equals one value $\mathbf{x}$ with probability λ, and another value $\mathbf{x}'$ with probability $1 - \lambda$. Jensen's inequality states the more general form for an arbitrary distribution on $\mathbf{X}$.

2 Convex Optimization

- In numerous fields, we are frequently interested in minimizing some cost function (or maximizing some utility function), possibly subject to certain constraints (see below for a variety of examples).

- Consider the following general form of optimization problem:

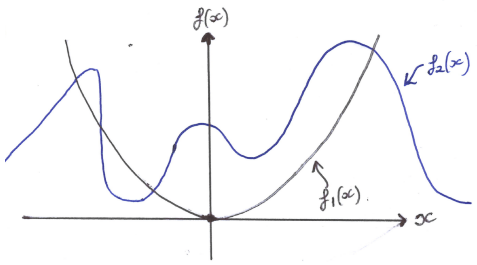
$$\begin{aligned} &\text{minimize}_{\mathbf{x}} && f_0(\mathbf{x}) && && (1)\\ &\text{subject to} && f_i(\mathbf{x}) \leq 0, && i = 1, \ldots, m_{\text{ineq}} \\ & && h_i(\mathbf{x}) = 0, && i = 1, \ldots, m_{\text{eq}}. \end{aligned}$$

  There are m_{ineq} *inequality constraints* and m_{eq} *equality constraints*.

- **Definition.** We say that (1) is a *convex optimization problem* if (i) $f_0(\mathbf{x})$ is convex; (ii) $f_i(\mathbf{x})$ is convex for all $i = 1, \ldots, m_{\text{ineq}}$; (iii) $h_i(\mathbf{x})$ is affine for all $i = 1, \ldots, m_{\text{eq}}$.

- This definition is very useful because, although solving (constrained or unconstrained) optimization problems is extremely hard in general, convexity is usually enough to permit finding a solution (sometimes analytically, but more often numerically).

- We can get some intuition by looking at the 1D case – which of these functions is easier to optimize using gradient descent techniques?

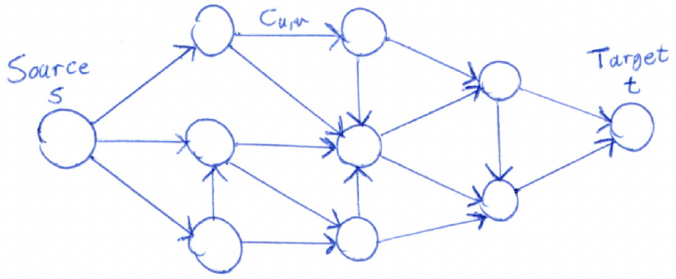


3 Examples of Convex Optimization Problems

Convexity may initially seem like a property that is rarely encountered in practice, but in fact it is surprisingly widespread and far-reaching. Just a few examples are given below.

Maximum flow via linear programming:

- Consider a directed graph with a number of nodes, two of which are the *source* and *target* (the latter is also known as the sink or destination):



- Let V denote the set of nodes and E denote the set of edges in the graph (i.e., $(u, v) \in E$ indicates an edge pointing from u to v), and let s and t denote the source and target nodes.

- Suppose that each edge $(u, v) \in E$ has a capacity c_{uv}, and consider the problem of creating as much "flow" (e.g., of information) from the source to target without exceeding any edge's capacity constraint.
- This naturally gives rise to a linear program: Letting f_{uv} represent the flow from u to v,

$$\begin{aligned} \text{maximize}_{\{f_{uv}\}_{(u,v)\in E}} \quad & \sum_{v\,:\,(s,v)\in E} f_{sv} \\ \text{subject to} \quad & 0 \le f_{uv} \le c_{uv} \quad \forall (u,v) \in E \\ & \sum_{u\,:\,(u,v)\in E} f_{uv} = \sum_{w\,:\,(v,w)\in E} f_{vw} \quad \forall v \in (V \setminus \{s,t\}), \end{aligned} \tag{2}$$

  where the first constraint simply constrains the total flow through each edge to be non-negative and not exceed capacity, and the second one constrains the total flow into any node to equal the total flow out (except for the source and target). The objective is to maximize the flow coming out of the source (which, by the constraints imposed, matches the amount coming into the target).
- Lagrange duality (covered below) for this problem is closely related to the famous *max-flow min-cut theorem* – we briefly explore this in an example later.

Estimation/regression via loss minimization:

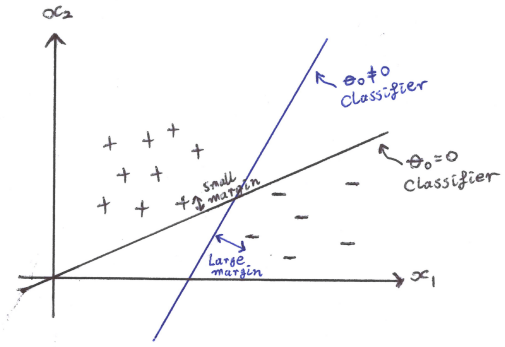
- In the problem of linear regression, one is given n examples $(\mathbf{x}_i, y_i)$ (for $i = 1, \ldots, n$) with $\mathbf{x}_i \in \mathbb{R}^n$ and $y_i \in \mathbb{R}$, and seeks to find linear weights $\boldsymbol{\theta}$ such that $\boldsymbol{\theta}^T \mathbf{x}_i$ approximates y_i well for all i (and similarly for unseen $(\mathbf{x}, y)$ pairs).
- A famous approach is *least squares*:

$$\text{minimize}_{\boldsymbol{\theta}} \sum_{i=1}^{n} (y_i - \boldsymbol{\theta}^T \mathbf{x}_i)^2,$$

  which is an unconstrained convex optimization problem (and a rare example of one that has an explicit closed-form solution). Other convex functions can also be used, e.g., using $|y_i - \boldsymbol{\theta}^T \mathbf{x}_i|$ instead of $(y_i - \boldsymbol{\theta}^T \mathbf{x}_i)^2$ provides better robustness to outliers.
- Constraints can also naturally enter, e.g., we may have prior information that the weights sum to one (so constrain $\sum_{i=1}^d \theta_i = 1$) or lie within a ball of a certain radius r (so constrain $\sum_{i=1}^d \theta_i^2 \le r$).

Maximum margin classification:

- The goal of binary classification seeks to construct a classifier that can reliably separate one class from another (e.g., distinguish spam vs. non-spam emails).
- Similarly to regression, we can represent inputs by vectors $\mathbf{x}_i \in \mathbb{R}^d$, and let $y_i \in \{-1, 1\}$ denotes its binary label (unlike regression where y was a general real value). Suppose that we are given a dataset with n such pairs, i.e., $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$.
- The class of *linear classifiers* takes the form $\hat{y} = \text{sign}(\boldsymbol{\theta}^T \mathbf{x} + \theta_0)$ for some weights $\boldsymbol{\theta} \in \mathbb{R}^d$ and $\theta_0 \in \mathbb{R}$. (Here θ_0 is an extra *offset* that was omitted in the regression example.)
- When the data set being learned from is perfectly separable, a natural approach is to find the classifier with the *largest margin* (i.e., maximize the distance between the margin and the closest data point):



- Using some geometric analysis and manipulation, this turns out to be a convex optimization problem:

$$\text{minimize}_{\boldsymbol{\theta},\theta_0} \quad \frac{1}{2}\|\boldsymbol{\theta}\|^2 \quad \text{subject to} \quad y_i(\boldsymbol{\theta}^T \mathbf{x}_i + \theta_0) \geq 1, \quad \forall i = 1, \ldots, n. \tag{3}$$

  (See Lectures 2 and 6a of https://www.comp.nus.edu.sg/~scarlett/CS5339_notes/ for further details.) Lagrange duality for this problem will be discussed briefly below.

Power allocation in communication:

- A famous formula in information theory gives the maximum possible rate of communication over a communication channel that adds Gaussian noise to its input: $R = \frac{1}{2}\log\left(1+\frac{P}{\sigma^2}\right)$, where P is the input power and σ^2 is the noise variance.
  - Note: Due to the (very slow) sub-linear growth of the $\log(\cdot)$ function, this means that we have *diminishing returns* as we increase the power level P
- Now imagine that we have K communication channels with different noise levels $\sigma_1^2, \ldots, \sigma_K^2$, and the constraint is on the *total power*: $P_1 + \ldots + P_K \leq P_{\text{total}}$. How should we allocate power to maximize the total rate $R_1 + \ldots + R_K$?
- This is a concave maximization problem:

$$\begin{aligned} \text{maximize}_{P_1,\ldots,P_K} \quad & \sum_{i=1}^{K} \frac{1}{2}\log\left(1+\frac{P_i}{\sigma_i^2}\right) \\ \text{subject to} \quad & \sum_{i=1}^{K} P_i \leq P_{\text{total}}, \\ & P_i \geq 0, \qquad i = 1, \ldots, K. \end{aligned} \tag{4}$$

- Lagrange dual analysis (covered below) gives rise to a well-known solution termed *waterfilling*.

Portfolio optimization:

- Here we give one simple example from finance applications, known as Markowitz portfolio design.
- Suppose that we have a model for price changes in the form of a distribution on a random vector $\mathbf{p} \in \mathbb{R}^d$, where d is the number of assets (e.g., stocks) we can invest in, and p_i is the random price change for asset i. Specifically, let $\mu_{\mathbf{p}} \in \mathbb{R}^d$ denote the mean vector of $\mathbf{p}$, and $\Sigma_{\mathbf{p}} \in \mathbb{R}^{d\times d}$ denote

the covariance matrix (i.e., the diagonal entries give the variances, and the off-diagonals capture the correlations between price changes of different assets).

- Suppose that we would be satisfied with any average return of $r_{\min}$ or higher, but we are risk-averse so want to keep the variance low. Based on this idea, the Markowitz portfolio design is as follows:

$$\begin{aligned} \text{minimize}_{\mathbf{x}\in\mathbb{R}^d} \quad & \mathbf{x}^T\Sigma_{\mathbf{p}}\mathbf{x} \\ \text{subject to} \quad & \mu_{\mathbf{p}}^T\mathbf{x} \geq r_{\min} \\ & \sum_{i=1}^{n} x_i = 1, \end{aligned}$$

  where $\mathbf{x} = [x_1, \ldots, x_d]^T$ with x_i being the proportion invested into asset i. The last constraint arises from the total proportion being 1; constraining $\mu_{\mathbf{p}}^T\mathbf{x} \geq r_{\min}$ captures the goal of attaining the desired average return (or higher); and minimizing $\mathbf{x}^T\Sigma_{\mathbf{p}}\mathbf{x}$ captures the goal of lowering variance.

- Once again, this is a convex optimization problem.

4 Lagrange Multipliers and Duality

- Warm-up: We can get some rough intuition behind Lagrange multipliers by consider the two-function case: Minimize $f(\mathbf{x})$ subject to $g(\mathbf{x}) \leq 0$. Suppose both are differentiable. Let $\mathbf{x}^*$ be a point that we believe to be a minimizer.

  If we could find a direction $\mathbf{v}$ such that $\mathbf{v}^T\nabla g(\mathbf{x}^*) = \mathbf{0}$ but $\mathbf{v}^T\nabla f(\mathbf{x}^*) \neq \mathbf{0}$, then (*a bit informally / hand-wavey*) we could move a tiny amount in some direction to decrease f while still satisfying the constraint on g. If $\mathbf{x}^*$ is a minimizer then this should be impossible, and thus either (i) $\nabla f(\mathbf{x}^*) = \mathbf{0}$, or (ii) ∇f and ∇g are parallel,[5] i.e., $\nabla f(\mathbf{x}^*) + \lambda\nabla g(\mathbf{x}^*) = \mathbf{0}$. Case (i) may occur when the constraint is "inactive" (i.e., removing it makes no difference), whereas Case (ii) gives a different type of optimality condition stating that $\mathbf{x}^*$ minimizes $L(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda g(\mathbf{x})$ for suitably-chosen λ (*Lagrange multiplier*).

- We proceed with a formal treatment of the general case. For an optimization problem of the form (1), the *Lagrangian* is defined as

$$L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) = f_0(\mathbf{x}) + \sum_{i=1}^{m_{\text{ineq}}} \lambda_i f_i(\mathbf{x}) + \sum_{i=1}^{m_{\text{eq}}} \nu_i h_i(\mathbf{x}), \tag{5}$$

  where we have introduced extra parameters $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_{m_{\text{ineq}}})$ and $\boldsymbol{\nu} = (\nu_1, \ldots, \nu_{m_{\text{eq}}})$. These are known as *Lagrange multipliers*.

  - We assume that $\lambda_i \geq 0$ for all i, whereas $\nu_i \in \mathbb{R}$ may be positive or negative.
  - Intuition: We no longer insist that $f_i(\mathbf{x}) \leq 0$, but we pay a penalty (scaled by λ_i) if it fails to hold. Conversely, we are "rewarded" if $f_i(\mathbf{x}) < 0$, i.e., strict inequality.

- **Important observation.** For any $\mathbf{x}$ feasible in (1), and any $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ with $\lambda_i \geq 0$, we have

$$L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) \leq f_0(\mathbf{x}). \tag{6}$$

[5] Two non-zero vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ are parallel (i.e., one is a multiple of the other) if and only if, for all $\mathbf{u}$, the inner products $\mathbf{u}^T\mathbf{v}_1$ and $\mathbf{u}^T\mathbf{v}_2$ are either both zero or both non-zero.

- Proof: Follows immediately from $\lambda_i \geq 0$, $f_i(\mathbf{x}) \leq 0$, and $h_i(\mathbf{x}) = 0$.

- Minimizing both sides of (6) over $\mathbf{x}$ gives

$$\min_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) \leq f_0(\mathbf{x}^*), \tag{7}$$

  where $\mathbf{x}^*$ is an optimal solution to (1).

  - The function

$$g(\boldsymbol{\lambda}, \boldsymbol{\nu}) = \min_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu})$$

    is called the *Lagrange dual function.*

- Since $g(\boldsymbol{\lambda}, \boldsymbol{\nu})$ lower bounds $f_0(\mathbf{x}^*)$ according to (7), it is natural to look for the *best (highest) lower bound*. This leads to the *Lagrange dual problem*:

$$\begin{aligned} &\text{maximize}_{\boldsymbol{\lambda}, \boldsymbol{\nu}} && g(\boldsymbol{\lambda}, \boldsymbol{\nu}) \\ &\text{subject to} && \lambda_i \geq 0, \qquad i = 1, \ldots, m_{\text{ineq}}. \end{aligned} \tag{8}$$

  Henceforth, let $(\boldsymbol{\lambda}^*, \boldsymbol{\nu}^*)$ denote the maximizer.

- **Duality.**

  - Since (7) holds for all $(\boldsymbol{\lambda}, \boldsymbol{\nu})$, it holds in particular for $(\boldsymbol{\lambda}^*, \boldsymbol{\nu}^*)$, yielding

$$g(\boldsymbol{\lambda}^*, \boldsymbol{\nu}^*) \leq f_0(\mathbf{x}^*).$$

    This is known as *weak duality.*

  - One of the most important results in convex optimization is that, if the original optimization problem is convex (i.e., f_0 and f_i are convex functions, and h_i is are linear functions), and a mild regularity condition holds, then

$$g(\boldsymbol{\lambda}^*, \boldsymbol{\nu}^*) = f_0(\mathbf{x}^*). \tag{9}$$

    This is known as *strong duality.*

    * There are many possible "mild regularity conditions"; the most well-known is known as *Slater's condition*: There exists at least one feasible point $\mathbf{x}$ satisfying the constraints of (1) with strict inequality (i.e., $f_i(\mathbf{x}) < 0$ and $h_i(\mathbf{x}) = 0$).
    * Another (more restrictive) sufficient condition is that the constraint functions f_i ($i = 1, \ldots, m_{\text{ineq}}$) are not only convex, but linear (and a feasible point exists).

  - Minimax theorem viewpoint: One way to understand duality is to interpret the original constrained optimization problem as solving

$$\min_{\mathbf{x}} \max_{\boldsymbol{\lambda} \geq \mathbf{0}, \boldsymbol{\nu}} L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}).$$

    This is because the inner maximization (more precisely a supremum) equals ∞ whenever $f_i(\mathbf{x}) > 0$ or $h_i(\mathbf{x}) \neq 0$, because any arbitrarily large value can be achieved by taking the corresponding λ_i

or ν_i to be huge. In addition, when $\mathbf{x}$ satisfies the constraints (i.e., each $f_i(\mathbf{x}) \leq 0$ and $h_i(\mathbf{x}) = 0$), it is not hard to show that that $\max_{\boldsymbol{\lambda} \geq \mathbf{0}, \boldsymbol{\nu}} L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) = f_0(\mathbf{x})$ (achieved by $\boldsymbol{\lambda} = \mathbf{0}$ and $\boldsymbol{\nu} = \mathbf{0}$).

In contrast, the Lagrange dual problem solves

$$\max_{\boldsymbol{\lambda} \geq \mathbf{0}, \boldsymbol{\nu}} \min_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}).$$

So almost everything is the same – just the max and min are swapped!

It a well-known fact of optimization and game theory that $\min_A \max_B f(A, B) \geq \max_B \min_A f(A, B)$. Strong duality is related to the *minimax theorem* (see https://en.wikipedia.org/wiki/Minimax_theorem), which states that in fact

$$\min_A \max_B f(A, B) = \max_B \min_A f(A, B)$$

in the case that $f(A, \cdot)$ is concave in B, $f(\cdot, B)$ is convex in A, and some other "mild" conditions hold (e.g., $\min_A$ and $\max_B$ are optimizations over compact sets).

- Optimality certificate viewpoint: Notice that any primal feasible $\mathbf{x}$ provides an *upper bound* to the optimal solution. Correspondingly, any dual feasible $(\boldsymbol{\lambda}, \boldsymbol{\mu})$ provides a *lower bound* to the optimal solution. When strong duality holds, we can get *matching upper and lower bounds*, which provide us a proof of optimality, so we call this a *certificate*.
- Other viewpoints (**Optional**): See Section 5 Boyd/Vandenberghe for other interpretations, including geometric (Section 5.3; see also https://www.argmin.net/p/ends-in-a-draw for a summary) and sensitivity analysis based (Section 5.6). In the latter, some statements are given on how the value of the Lagrange multiplier relates to how much the optimal value changes upon tightening/loosening the constraints.

- **Example 1 (Linear programming).**
  - Consider a linear program of the form

$$\begin{aligned} \text{maximize}_{\mathbf{x}} \quad & \mathbf{c}^T \mathbf{x} && (10) \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} = \mathbf{b},\ \mathbf{x} \geq \mathbf{0} && (11) \end{aligned}$$

    for some matrix $\mathbf{A} \in \mathbb{R}^{m \times d}$ and vectors $\mathbf{b} \in \mathbb{R}^m$ and $\mathbf{c} \in \mathbb{R}^d$. The inequality $\mathbf{x} \geq \mathbf{0}$ should be interpreted as holding element-wise.
  - Interpreting this as being in the form (1) with $m_{\text{ineq}} = m$ and $m_{\text{eq}} = d$, we have the Lagrangian

$$\begin{aligned} L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) &= -\mathbf{c}^T \mathbf{x} - \sum_{i=1}^{d} \lambda_i x_i + \sum_{i=1}^{m} \nu_i (\mathbf{a}_i^T \mathbf{x} - b_i) \\ &= -\mathbf{c}^T \mathbf{x} - \boldsymbol{\lambda}^T \mathbf{x} + \boldsymbol{\nu}^T (\mathbf{A}\mathbf{x} - \mathbf{b}) \\ &= -\mathbf{b}^T \boldsymbol{\nu} + (\mathbf{A}^T \boldsymbol{\nu} - \boldsymbol{\lambda} - \mathbf{c})^T \mathbf{x}, \end{aligned}$$

    where $\mathbf{a}_i$ is the i-th row of $\mathbf{A}$, b_i is the i-th entry of $\mathbf{b}$, etc.
    - Note: Switching from "maximize" to "minimize" requires taking $f_0(\mathbf{x}) = -\mathbf{c}^T \mathbf{x}$. Similarly, we rewrite $\mathbf{x} \geq \mathbf{0}$ as $-\mathbf{x} \leq \mathbf{0}$.

– Minimizing over $\mathbf{x}$, we find that $g(\boldsymbol{\lambda}, \boldsymbol{\nu})$ (which we recall is $\min_{\mathbf{x}} L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\nu})$) takes the form

$$g(\boldsymbol{\lambda}, \boldsymbol{\nu}) = \begin{cases} -\mathbf{b}^T \boldsymbol{\nu} & \mathbf{A}^T \boldsymbol{\nu} - \boldsymbol{\lambda} - \mathbf{c} = \mathbf{0} \\ -\infty & \text{otherwise.} \end{cases}$$

This is because whenever $\mathbf{A}^T \boldsymbol{\nu} + \boldsymbol{\lambda} + \mathbf{c} \neq 0$, one can just make a suitable entry of x_i arbitrarily large in either the positive or negative direction.

– Substituting this expression for $g(\boldsymbol{\lambda}, \boldsymbol{\nu})$ into (8) yields the *dual problem*:

$$\begin{aligned} \text{maximize}_{\boldsymbol{\lambda}, \boldsymbol{\nu}} \quad & -\mathbf{b}^T \boldsymbol{\nu} \\ \text{subject to} \quad & \boldsymbol{\lambda} \geq \mathbf{0}, \\ & \mathbf{A}^T \boldsymbol{\nu} - \boldsymbol{\lambda} - \mathbf{c} = \mathbf{0}, \end{aligned}$$

where the second constraint can be introduced since all other values yield a (certainly suboptimal) value of $-\infty$. Since $\boldsymbol{\lambda}$ does not appear in the objective function, we can further simplify the above maximization to

$$\begin{aligned} \text{minimize}_{\boldsymbol{\nu}} \quad & \mathbf{b}^T \boldsymbol{\nu} \\ \text{subject to} \quad & \mathbf{A}^T \boldsymbol{\nu} \geq \mathbf{c}. \end{aligned}$$

– If we replace $\mathbf{Ax} = \mathbf{b}$ by $\mathbf{Ax} \leq \mathbf{b}$ in the original formulation, then we arrive at a similar dual expression but with the added constraint $\boldsymbol{\nu} \geq 0$.

– **An intuitive interpretation:**

* The original problem constrains $\mathbf{Ax} = \mathbf{b}$; multiplying both sides on the left by $\boldsymbol{\nu}^T$ gives $\boldsymbol{\nu}^T \mathbf{Ax} = \boldsymbol{\nu}^T \mathbf{b}$, or equivalently $(\mathbf{A}^T \boldsymbol{\nu})^T \mathbf{x} = \mathbf{b}^T \boldsymbol{\nu}$ (by standard properties of the transpose)
* Now, since $\mathbf{x} \geq \mathbf{0}$ and we are maximizing $\mathbf{c}^T \mathbf{x}$, we find that if $\mathbf{A}^T \boldsymbol{\nu} \geq \mathbf{c}$, it holds that $(\mathbf{A}^T \boldsymbol{\nu})^T \mathbf{x}$ is at least as high as $\mathbf{c}^T \mathbf{x}$. Then, by the previous dot point, $\mathbf{b}^T \boldsymbol{\nu}$ is at least as high as $\mathbf{c}^T \mathbf{x}$.
* Hence, for any $\boldsymbol{\nu}$ that satisfies $\mathbf{A}^T \boldsymbol{\nu} \geq \mathbf{c}$, we have that $\mathbf{b}^T \boldsymbol{\nu}$ is at least as high as the original problem's optimal value, i.e., it is an *upper bound* to the optimal value.
* By minimizing over all such $\boldsymbol{\nu}$ (as is done in the dual expression), we are finding the *lowest (best) possible upper bound*, and this turns out to make the upper bound hold with equality.

• **Example 2 (Maxflow-mincut duality).**

– A famous theorem for the maxflow problem is *maxflow-mincut duality*: The value of the maximum flow is the same as the value of the minimum cut (i.e., the minimum total weight of edges whose removal makes the target unreachable from the source).

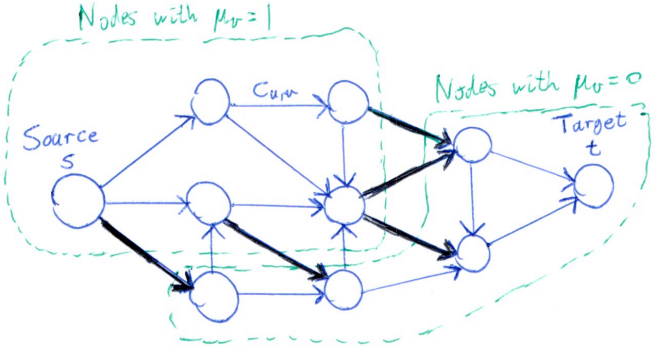
– A Lagrange dual analysis of (2) similar to Example 1 gives rise to the following:

$$\begin{aligned} \text{maximize}_{\{\lambda_{uv}\}_{(u,v)\in E}, \{\mu_v\}_{v \in V}} \quad & \sum_{(u,v)\in E} c_{uv} \lambda_{uv} \\ \text{subject to} \quad & \mu_s = 1, \mu_t = 0, \\ & \lambda_{uv} \geq \mu_u - \mu_v, \ \lambda_{uv} \geq 0 \quad (\forall (u,v) \in E). \end{aligned}$$

* Each λ_{uv} arises as a Lagrange multiplier for the edge's capacity constraint, and each μ_v arises as a Lagrange multiplier for the node's inflow=outflow constraint (except s and t, which is why their μ_v values are instead fixed to 1 and 0).

- Note that once the μ_v's are all specified, the λ_{uv}'s are trivially given by $\lambda_{uv} = \max\{0, \mu_u - \mu_v\}$. So the problem "minimize over all μ_v and λ_{uv}" is essentially just one of "minimize over all μ_v".
- If we additionally constrain $\mu_v \in \{0, 1\}$ and $\lambda_{uv} \in \{0, 1\}$ for all u and v, then the mincut interpretation is immediate: The nodes with $\mu_u = 1$ are on the "source side", those with $\mu_v = 0$ are on the "target side", and the goal is to minimize the sum of c_{uv} over all edges from the former to the latter (for which $\lambda_{uv} = 1$).

  An illustration (with bold edges being the 'cut' ones, whose c_{uv} values are summed):



- In other words, the Lagrange dual analysis has given a *linear programming relaxation* of the min-cut problem. But it turns out to be a case where the relaxation is *tight* – it gives the same answer as the integer-constrained problem. (See the next lecture for more on this.)

- **(**Optional**) Example 3 (Support vector machine).**
  - As given in the above examples, the maximum-margin hyperplane problem in classification can be cast as

$$\text{minimize}_{\boldsymbol{\theta},\theta_0} \quad \frac{1}{2}\|\boldsymbol{\theta}\|^2 \quad \text{subject to} \quad y_i(\boldsymbol{\theta}^T \mathbf{x}_i + \theta_0) \geq 1, \quad \forall i = 1, \ldots, n. \tag{12}$$

    This is a convex optimization problem with affine constraints, so strong duality holds.
  - By a similar analysis Example 1, the following dual optimization problem can be derived:

$$\begin{aligned} \text{maximize}_{\boldsymbol{\alpha}} \quad & \sum_{i=1}^n \alpha_t - \frac{1}{2}\sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \\ \text{subject to} \quad & \alpha_i \geq 0 \quad \forall i \in \{1, \ldots, n\}, \\ & \sum_{i=1}^n \alpha_i y_i = 0. \end{aligned}$$

    * Each $\alpha_i \geq 0$ arises as a Lagrange multiplier corresponding to the i-th data point's constraint, and the condition $\sum_{i=1}^n \alpha_i y_i = 0$ is related to an optimization over θ_0 (if θ_0 is removed from the problem altogether, then so is this constraint in the dual).
  - In addition, the Lagrange duality analysis reveals that $\boldsymbol{\theta} = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i$, and complementary slackness (covered below) can used to derive $\theta_0 = \frac{1}{y_t} - \boldsymbol{\theta}^T \mathbf{x}_t$ for any t such that $\alpha_t > 0$.

– A significant advantage of the dual formulation is that it depends on the $\mathbf{x}$'s only via inner products (e.g., $\mathbf{x}_i^T \mathbf{x}_j$), which allows for an application of the *kernel trick* where each inner product is replaced by a more general function $k(\mathbf{x}_i, \mathbf{x}_j)$. The idea is that we could "change the representation" of the input by replacing each $\mathbf{x}$ by some function $\phi(\mathbf{x})$, but the kernel trick allows us to do so *implicitly* instead of explicitly – there are many spaces where $\phi(\mathbf{x})$ is "complicated" (e.g., infinite-dimensional) and yet $\phi(\mathbf{x})^T \phi(\mathbf{x}')$ is "simple" (e.g., can be evaluated in linear time).

– See Lecture 6a of https://www.comp.nus.edu.sg/~scarlett/CS5339_notes/ for the complete details of this example.

5 The Karush-Kuhn-Tucker (KKT) Conditions

- In the case that strong duality holds as per (9), we have the following chain of inequalities:

$$\begin{aligned} f_0(\mathbf{x}^*) &= g(\boldsymbol{\lambda}^*, \boldsymbol{\nu}^*) \\ &= \min_{\mathbf{x}} \left\{ f_0(\mathbf{x}) + \sum_{i=1}^{m_{\text{ineq}}} \lambda_i^* f_i(\mathbf{x}) + \sum_{i=1}^{m_{\text{eq}}} \nu_i^* h_i(\mathbf{x}) \right\} \\ &\leq f_0(\mathbf{x}^*) + \sum_{i=1}^{m_{\text{ineq}}} \lambda_i^* f_i(\mathbf{x}^*) + \sum_{i=1}^{m_{\text{eq}}} \nu_i^* h_i(\mathbf{x}^*) \\ &\leq f_0(\mathbf{x}^*), \end{aligned}$$

  where we first applied the definition of g, then upper bounded the minimum by the specific value $\mathbf{x}^*$, then used the fact that $f_i(\mathbf{x}^*) \leq 0$ and $h_i(\mathbf{x}^*) = 0$.

- Since we ended up with $f_0(\mathbf{x}^*) \leq f_0(\mathbf{x}^*)$, both of the inequalities must hold with equality. Let's look at these in more detail:

  – The first inequality holding with equality gives

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} f_0(\mathbf{x}) + \sum_{i=1}^{m_{\text{ineq}}} \lambda_i^* f_i(\mathbf{x}) + \sum_{i=1}^{m_{\text{eq}}} \nu_i^* h_i(\mathbf{x}).$$

    Assuming the functions are differentiable, the fact that $\mathbf{x}^*$ is a minimizer means that the derivative must vanish:

$$\nabla f_0(\mathbf{x}^*) + \sum_{i=1}^{m_{\text{ineq}}} \lambda_i^* \nabla f_i(\mathbf{x}^*) + \sum_{i=1}^{m_{\text{eq}}} \nu_i^* \nabla h_i(\mathbf{x}^*) = \mathbf{0}.$$

  – The second inequality holding with equality gives

$$\lambda_i^* f_i(\mathbf{x}^*) = 0, \qquad i = 1, \ldots, m_{\text{ineq}}.$$

    This means that either $f_i(\mathbf{x}^*) = 0$ (i.e., the constraint holds with equality) or $\lambda_i^* = 0$. This property is known as *complementary slackness*.

- Summarizing the above leads to a set of conditions on $(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\nu}^*)$ known as the *KKT conditions*:

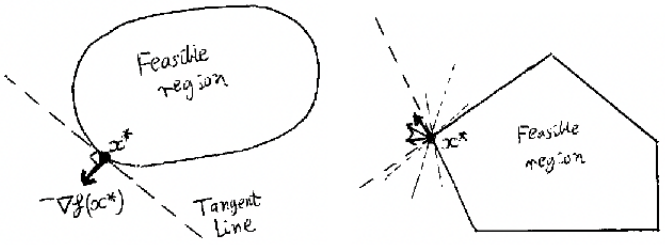
  1. (Primal feasibility) $f_i(\mathbf{x}^*) \leq 0$ for $i = 1, \ldots, m_{\text{ineq}}$, and $h_i(\mathbf{x}^*) = 0$ for $i = 1, \ldots, m_{\text{eq}}$.

2. (Dual feasibility) $\lambda_i^* \geq 0$ for $i = 1, \ldots, m_{\text{ineq}}$.
3. (Complementary slackness) $\lambda_i^* f_i(\mathbf{x}^*) = 0$ for $i = 1, \ldots, m_{\text{ineq}}$.
4. (Vanishing gradient) $\nabla f_0(\mathbf{x}^*) + \sum_{i=1}^{m_{\text{ineq}}} \lambda_i^* \nabla f_i(\mathbf{x}^*) + \sum_{i=1}^{m_{\text{eq}}} \nu_i^* \nabla h_i(\mathbf{x}^*) = \mathbf{0}$.

These generalize the requirement that the *unconstrained* maximizer of $f_0(\mathbf{x})$ should satisfy $\nabla f_0(\mathbf{x}^*) = 0$.

- General case: If strong duality holds, it is necessary that $(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\nu}^*)$ satisfy the KKT conditions.
- Convex case: If strong duality holds *and the primal problem is convex*, then $(\mathbf{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\nu}^*)$ satisfying the KKT conditions are *also sufficient for optimality* (the proof of this is omitted).

- **(**Optional**)** As a final note (without proof), *geometric* optimality conditions can also be formed: For constraint sets with smooth boundaries,the negative gradient vector $-\nabla f(\mathbf{x}^*)$ points perpendicular to the constraint set boundary (left figure below). A similar statement can be made for "pointy" constraint sets like polyhedra, except that we get an *entire cone* of possible directions (right figure below).



CS5275 Lecture 5: Submodularity

Jonathan Scarlett

February 18, 2025

Useful references:

- Krause and Golovin, "Submodular Function Maximization"
- Columbia lecture notes: `https://www.columbia.edu/~yf2414/ln-submodular.pdf`
- Various research papers linked throughout the sections below

Categorization of material:

- Core material: Sections 1–3 except the parts labeled "Optional'.
- Extra material: Section 4 (variations of greedy).

(Exam will strongly focus on "Core". Take-home assessments may occasionally require consulting "Extra".)

1 Introduction

In discrete optimization, it is common to encounter problems that consist of *choosing elements from a set*:

- Choosing items in resource allocation and similar problems (e.g., knapsack);
- Choosing subsets of nodes in graph problems (e.g., minimum cut);
- Choosing representative data points in data summarization;
- etc.

In this lecture, we interpret such problems as optimizing some $f(S)$ over $S \subseteq \{1, \dots, n\}$. Although we will not use it much, this could also equivalently be viewed as optimizing $f(\mathbf{x})$ over $\mathbf{x} \in \{0,1\}^n$; the equivalence is seen by simply interpreting S as $\{i : x_i = 1\}$.

Set function maximization: Find "best" subset S of ground set $V = \{1, \dots, n\}$.

- Unconstrained:

$$\text{maximize}_{S \subseteq V} \; f(S)$$

- Constrained:

$$\text{maximize}_{S \in \mathcal{S}} \; f(S)$$

  where the constraint set $\mathcal{S}$ contains subsets of V. We will especially focus on the constraint set $\mathcal{S} = \{S : |S| \leq k\}$ that enforces choosing at most k elements for some $k > 0$.

Much like continuous functions, assumptions are needed on f to give us hope of being able to perform optimization efficiently. We will consider the following two main assumptions, with an emphasis on the second one (submodularity).

Definition. A set function f is said to be *monotone* if, for all $S \subseteq T \subseteq V$, it holds that $f(S) \leq f(T)$.

- Note: When we need to specifically emphasize the direction of monotonicity, we will use the terminology *non-decreasing* or *non-increasing*. But unless explicitly specified otherwise, the term *monotone* will mean non-decreasing in this lecture.

Definition. A real-valued set function f is said to be *submodular* if, for all $S \subseteq T \subseteq V$ and $e \in V \backslash T$, it holds that

$$\Delta(e|S) \geq \Delta(e|T),$$

where we define $\Delta(e|S) = f(S \cup \{e\}) - f(S)$ (i.e., the gain of adding e to S), and similarly for $\Delta(e|T)$.

- Intuition: "Diminishing returns" – adding to a smaller set gains you more. This intuition is especially useful for monotone functions (it can get a bit less easy to picture in the general case).

- Related notions: (i) The function is *modular* if it always holds that

$$\Delta(e|S) = \Delta(e|T)$$

  It is not hard to see that this is equivalent to "linear", say $f(S) = \sum_{i \in S} c_i$, which is equivalent to $c^T x$ with $x \in \{0,1\}^n$ and $S = \{i : x_i = 1\}$.

  (ii) The function is *supermodular* if it always holds that

$$\Delta(e|S) \leq \Delta(e|T).$$

  We will focus on submodularity in this lecture. The types of problems we will solve below will be fairly trivial for modular (linear) functions, e.g., solved by sorting n numbers and taking the k highest ones.

Equivalent definitions of submodularity. For any function $f : 2^V \to \mathbb{R}$, all of the following are equivalent:

(i) $\Delta(e|S) \geq \Delta(e|T)$ for all $S \subseteq T$ and all $e \in V \setminus T$ (this is a repeat of the above definition)

(ii) $f(S) + f(T) \geq f(S \cup T) + f(S \cap T)$ for all S, T

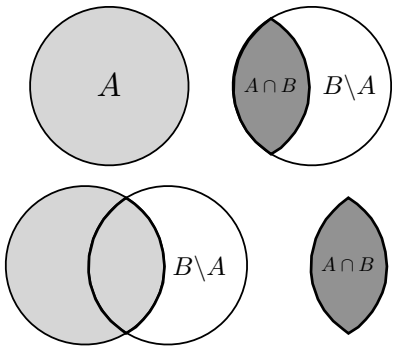
(iii) $\Delta(e|S) \geq \Delta(e|S \cup \{e'\})$ for all sets S and elements e, e'

(iv) (In the case that f is monotone) $f(T) \leq f(S) + \sum_{e \in T \setminus S} \Delta(e|S)$ for all S, T

We will not prove all of these, but we proceed to give some intuition and some details as follows:

- Definition (iii) is just (i) applied to $T = S \cup \{e'\}$, and it is not difficult to show that the property for $|T| = |S| + 1$ implies the general case. Definition (iv) just states that the overall gain of adding $T \setminus S$ is no larger than the individual gains of its elements, which is natural given diminishing returns.

- Definition (ii) may seem less intuitive, but can be understood visually as comparing what happens when we add $T \setminus S$ to a smaller set ($S \cap T$) vs. a larger set (S) – see the figure below. By the diminishing returns property, the former should cause a higher increase in f.



- There are fairly elementary proofs of going from one definition to any other equivalent one. For example, the following analysis shows that (iii) implies (ii): Fix any S, T, let $W = S \cap T$, and denote $S \setminus T = \{j_1, \ldots, j_s\}$ and $T \setminus S = \{k_1, \ldots, k_t\}$. Then

$$
\begin{aligned}
& f(T) - f(S \cap T) \\
= \;& f(W \cup \{k_1, \ldots, k_t\}) - f(W) \qquad \text{(def. } W \text{ and } k_i\text{)} \\
= \;& \sum_{i=1}^{t} \Big(f(W \cup \{k_1, \ldots, k_i\}) - f(W \cup \{k_1, \ldots, k_{i-1}\}) \Big) \qquad \text{(telescoping)} \\
\geq \;& \sum_{i=1}^{t} \Big(f(W \cup \{k_1, \ldots, k_i, j_1\}) - f(W \cup \{k_1, \ldots, k_{i-1}, j_1\}) \Big) \qquad \text{(using (iii))} \\
\vdots \;& \\
\geq \;& \sum_{i=1}^{t} \Big(f(W \cup \{k_1, \ldots, k_i, j_1, \ldots, j_s\}) - f(W \cup \{k_1, \ldots, k_{i-1}, j_1, \ldots, j_s\}) \Big) \qquad \text{using (iii)} \\
= \;& \sum_{i=1}^{t} \Big(f(S \cup \{k_1, \ldots, k_i\}) - f(S \cup \{k_1, \ldots, k_{i-1}\}) \Big) \qquad \text{(definition of } W \text{ and } j_{(\cdot)}\text{)} \\
= \;& f(S \cup T) - f(S) \qquad \text{(telescoping).}
\end{aligned}
$$

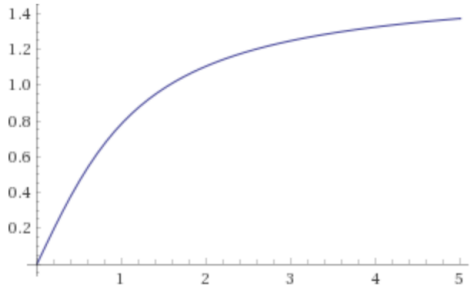
- As one more example, here is a proof that (i) implies (iv) when f is monotone: Again letting $T \setminus S =$

$\{k_1, \ldots, k_t\}$, we have

$$\begin{aligned} f(T) &\leq f(S \cup T) \qquad \text{(monotonicity)} \\ &= f(S) + (f(S \cup T) - f(S)) \\ &= f(S) + \sum_{i=1}^{t} \Big(f(S \cup \{k_1, \ldots, k_i\}) - f(S \cup \{k_1, \ldots, k_{i-1}\}) \Big) \qquad \text{(telescoping)} \\ &\leq f(S) + \sum_{i=1}^{t} (f(S \cup \{k_i\}) - f(S)). \qquad \text{(using (i))} \end{aligned}$$

Relations to convexity and concavity: Submodular functions share some properties of *concave functions*:

- Diminishing returns (see the figure below)



- (Non-monotone case) Any local maximum is within a factor of $\frac{1}{2}$ of globally optimal
- Maximization (constrained or unconstrained) can be done *approximately* using computationally efficient algorithms
- A function of the form $f(S) = g(|S|)$ is submodular if g is concave (see below for a more general statement).

...but they also share some properties with *convex functions*:

- Unconstrained minimization can be done *exactly* using computationally efficient algorithms (constrained not so!)
- A natural extension from sets (represented by $\{0,1\}$-values) to continuous values (i.e., $[0,1]$) called the *Lovász extension* is a convex function

Sometimes neither concave-like not convex-like properties apply (e.g., the pointwise max. or min. of two submodular functions may not be submodular). A classical paper on the above connections is "Submodular functions and convexity" (Lovász, 1983).

Properties of Submodular Functions: Let f_1 and f_2 be submodular functions. Then:

1. **Linear combinations:** If c_1, c_2 are positive, then $f(S) = c_1 f_1(S) + c_2 f_2(S)$ is submodular.
2. **Concave of modular:** If $g : 2^V \to \mathbb{R}$ is modular and $h : \mathbb{R} \to \mathbb{R}$ is concave, then $f(S) = h(g(S))$ is submodular. An important special case is $g(S) = |S|$.

- Intuition: At least for monotone g, this can be understood from diminishing returns

3. **Residual:** $f(S) = f_1(S \cup B) - f_1(B)$ is submodular for any B.
   - Intuition: This function measures how much S increases f_1 after the elements of B have already been chosen. The subtraction of $f_1(B)$ is optional, but ensures $f_1(\emptyset) = 0$.
4. **Conditioning:** $f(S) = f_1(S \cap A)$ is submodular for any A.
   - Intuition: Items from A get added as usual, but those outside A play no role.
5. **Reflection:** $f(S) = f_1(V \backslash S)$ is submodular.
   - Intuition: This is analogous to how if $g(x)$ is convex, so is $g(c - x)$.
6. **Truncation:** If f_1 is also monotone, then $f(S) = \min\{c, f_1(S)\}$ is submodular for any $c \in \mathbb{R}$.
7. **Minimum:** Although $f(S) = \min\{f_1(S), f_2(S)\}$ is not submodular in general, it is submodular when either $f_1 - f_2$ or $f_2 - f_1$ is monotone.

Some of these properties (e.g., linear combinations, residual, conditioning) can be proved essentially directly from the definition of submodularity. Here are a couple of proofs for the "less obvious" properties:

- Proof of reflection property: Fix any sets S and T, and let $W = V \backslash (S \cup T)$. Using the residual property (without the subtracted term, which we won't need), the function $g(A) = f_1(A \cup W)$ is submodular, so definition (ii) of submodularity gives

  $$g(A) + g(B) \geq g(A \cup B) + g(A \cap B).$$

  Now let $f(S) = f_1(V \backslash S)$ and consider the following:

  $$f(S) + f(T) = f_1(V \backslash S) + f_1(V \backslash T)$$
  $$f(S \cap T) + f(S \cup T) = f_1(V \backslash (S \cap T)) + f_1(V \backslash (S \cup T))$$

  Letting $A = S \setminus T$ and $B = T \setminus S$, it is straightforward to check via the above definitions that $f_1(V \backslash S) = g(B)$, $f_1(V \backslash T) = g(A)$, $f_1(V \backslash (S \cap T)) = g(A \cup B)$, and $f_1(V \backslash (S \cup T)) = g(A \cap B)$ (in fact $A \cap B = \emptyset$). Therefore, by the above 3 displayed equations, submodularity of f (i.e., $f(S) + f(T) \geq f(S \cup T) + f(S \cap T)$) is a consequence of submodularity of g.

- Proof of truncation property: Let $\Delta_1(e|S) = f_1(S \cup \{e\}) - f_1(S)$ and $\Delta_f(e|S) = f(S \cup \{e\}) - f(S)$, where $f(S) = \min\{c, f_1(S)\}$. We use the first definition of submodularity, stating that $\Delta_1(e|S) \geq \Delta_1(e|T)$ when $S \subseteq T$.

  We wish to show that $\Delta_f(e|S) \geq \Delta_f(e|T)$ as well. Notice that if $f(T) = c$ then adding further items will just keep the value at c (by monotonicity of f_1), giving $\Delta_f(e|T) = 0$ and making the desired inequality trivial. So we can assume $f(T) < c$, which implies $f(S) < c$ by monotonicity. Then we have

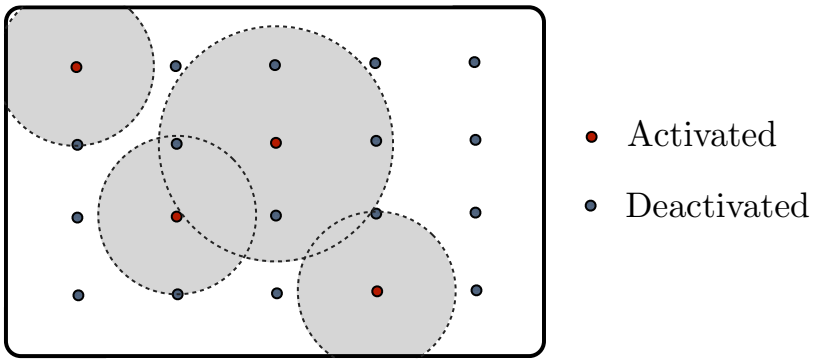
  $$\Delta_f(e|S) = f(S \cup \{e\}) - f(S) = \min\{c, f_1(S \cup \{e\})\} - f_1(S) = \min\{c - f_1(S), \Delta_e(e|S)\},$$

  and similarly for $\Delta_f(e|T)$. The desired inequality follows since $c - f_1(S) \geq c - f_1(T)$ (by monotonicity) and $\Delta_e(e|S) \geq \Delta_e(e|T)$ (by submodularity).

2 Examples of Submodular Functions

Example 1: Coverage functions

$$f(S) = \text{Area covered by activating all sensors in } S$$



- This is clearly monotone (activating more sensors implies more coverage) and submodular (the more sensors were activated already, the less gain there will be by activating another one)
- If we replace the continuous notion of "area" by the discrete notion of "cardinality", we get the *set cover* problem. A special case of set cover is *vertex cover*, which is the (NP-hard) problem of finding a set of vertices that includes at least one endpoint of every edge. This is related to maximizing $f(S)$, defined to be the number of edges connected to at least one node in S. (Specifically, we are interested in how small S can be to get $f(S) = |E|$, the total number of edges.)

Example 2: Let $\mathbf{X}$ be a matrix, let V be the indices of its columns, and let $\mathbf{X}_S$ be the submatrix formed by keeping only the columns indexed by S. Then $r(S) = \text{rank}(\mathbf{X}_S)$ is *monotone submodular*.

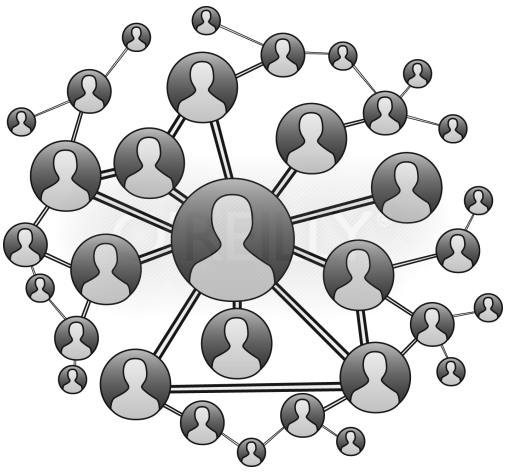
- Recall that rank can be interpreted as the maximum number of linearly independent columns (or rows).
- The intuition on monotonicity: Adding another column can either keep that maximum the same or increase it by one.
- The intuition on submodularity: Adding a column to a larger set means that there are more ways in which the column could already be a linear combination of the existing ones.

Example 3: Influence maximization

$$f(S) = \text{Total number of users influenced by advertising to } S$$

See the next page for an illustrative figure.

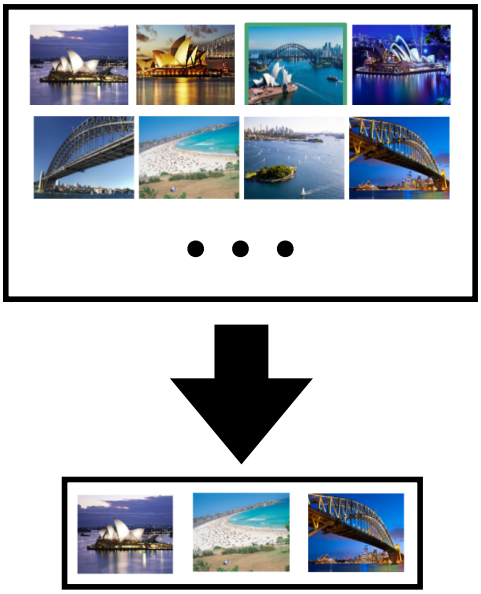
- There are several ways that influence could reasonably be measured (many of which lead to submodularity, but not all).



- Perhaps the simplest model is that given the graph $G = (V, E)$, advertising to some $v \in V$ amounts to *influencing* v *and its direct neighbors*. This leads to $f(S)$ being submodular in the same way as Example 1.
- There are much more sophisticated models, such as v influencing each of its neighbors independently with probability p, and each of those influenced people subsequently influencing their own neighbors with some (possibly smaller) probability, etc., still leading to submodularity if $f(S)$ is taken to be the *average* number of people influenced.
- See (e.g.,) the paper "Maximizing the spread of influence through a social network" for more on this application if you are interested.

Example 4: Data summarization

$$f(S) = \text{Representativeness of images in } S$$



- Again, there are many reasonable measures of "representativeness", some of which are submodular and some of which are not.
- For example, one way of measuring representativeness (and indeed giving submodularity) is to try to make every image in the full set V close to the nearest image in S:

$$g(S) = \frac{1}{n} \sum_{v \in V} \min_{s \in S} d(s, v),$$

  where $d(s, v)$ is a pre-specified distance measure between two images s and v. The minimizing s in the summation above is sometimes referred to as a *medoid*.
- The more representative S is, the smaller g is, so we could pose the data summarization problem as maximizing $f(S) = -g(S)$. To make this well-defined even when $S = \emptyset$, it's more common to further define a *reference element* e_0 and instead consider

$$f(S) = g(\{e_0\}) - g(S \cup \{e_0\}),$$

  which is monotone, submodular, and satisfies $f(\emptyset) = 0$ – see the appendix for a related exercise from which you can prove this.
- See (e.g.,) the paper "A class of submodular functions for document summarization" for more on this application if you are interested.

Example 5: Graph cut

- Given a graph $G = (V, E)$, split the nodes V into (S, S^c)
- Graph cut function $f(S)$: Number of edges between S and S^c
- This is *submodular* but *non-monotone* – see the appendix for the proof.
- Maximizing this $f(S)$ is the Maximum Cut (MAXCUT) problem, which is known to be NP-hard.

Example 6: The *entropy* of a random variable is defined as

$$H(X) = \sum_x P_X(x) \log \frac{1}{P_X(x)}. \tag{1}$$

Let $\mathbf{X} = (X_1, \ldots, X_n)$ be a random vector, and let $X_S = \{X_i\}_{i \in S}$. Then the entropy $f(S) = H(\mathbf{X}_S)$ is *monotone submodular*.

- Roughly speaking, $H(\mathbf{X}_S)$ measures the amount of uncertainty in the random variables indexed by S.
- Intuition on monotonicity: Adding more variables amounts to having higher uncertainty.
- Intuition on submodularity: Adding to a larger set increases the uncertainty less, because the existing variables provide information about the new variable (e.g., correlations or other dependencies).
- You may want to try proving monotonicity/submodularity after we have covered information theory.

3 Submodular Optimization

Several problems in theoretical computer science, game theory, machine learning, etc. can be cast as a submodular optimization problem.

Problem statement: Given a submodular function f and constraint set $\mathcal{S}$

$$\min_{S \in \mathcal{S}} f(S) \qquad \text{(SFMin)}$$

$$\max_{S \in \mathcal{S}} f(S) \qquad \text{(SFMax)}$$

If $\mathcal{S}$ contains all 2^n possible choices of S, the problem is said to be *unconstrained*.

For most of the lecture, we focus on the following:

- Maximization only
- Monotone functions only, typically "normalized" so that $f(\emptyset) = 0$
- Cardinality constraint: $\mathcal{S} = \{S : |S| \leq k\}$

Before doing so, we briefly mention some results regarding some of the other categories.

3.1 (**Optional**) Non-monotone submodular maximization

- Unconstrained SFMax admits a computationally efficient 1/2-*approximation* algorithm (i.e., an algorithm guaranteed to attain an f value within 1/2 of the highest possible). The factor 1/2 cannot be improved in general, in the sense that attaining a $(1/2 + \epsilon)$-approximation requires exponentially many function evaluations. See the research papers "A tight linear time (1/2)-approximation for unconstrained submodular maximization" and "Maximizing non-monotone submodular functions" for details.
- The cardinality-constrained (non-monotone) case has proved challenging; to my knowledge the best possible approximation constant is not known, but the paper "Submodular Maximization with Cardinality Constraints" shows (for non-negative f) that it is between $\frac{1}{e}$ and $\frac{1}{2}$, and equals $\frac{1}{2}$ in the limit of large n when the cardinality is $k = \frac{n}{2}$.
- Algorithms for this problem are also typically greedy-like but not quite as simple as the monotone setting (covered below), e.g., keeping track of two lists from which items are added/removed, using randomized methods for choosing additions and removals, or applying greedy steps to continuous extensions rather than directly on the discrete function.

3.2 (**Optional**) Submodular minimization

- Unconstrained submodular minimization turns out to be a problem that can be solved efficiently. A very brief outline is as follows:
  - The idea is to interpret $f(S)$ as $f(\mathbf{x})$, where $\mathbf{x} \in \{0,1\}^n$ (entry 1 if the element is in S, and 0 otherwise), then extend $f(\mathbf{x})$ to a function on $[0,1]^n$, minimize it "sufficiently accurately", and finally convert the solution back to a $\{0,1\}^n$-valued one.
  - The extension from $\{0,1\}^n$ to $[0,1]^n$ is done by defining $f(\mathbf{x}) = \mathbb{E}[f(S_\lambda(\mathbf{x}))]$ for $\lambda \sim \text{Uniform}[0,1]$, where $S_\lambda(\mathbf{x}) = \{i : x_i > \lambda\}$. This is called the *Lovász extension*.

- Crucially, the Lovász extension is a convex function, which is why it is "easy" to minimize.

- In contrast, *constrained* submodular minimization is very hard, with special cases including NP-hard problems such as densest k-subgraph and uniform graph partitioning (which we won't describe here). In fact, in general it is hard to even obtain a solution that is optimal to within a constant factor (e.g., at most 1000 times the optimal value).

3.3 Cardinality-Constrained Submodular Maximization

- The cardinality-constrained problem is NP-hard in general (e.g., this can be shown via the special case of vertex cover). But it turns out that a simply greedy algorithm gives good approximation guarantees.

- **Greedy algorithm:**

  1. Initialize $S_0 = \emptyset$
  2. For $i = 1, \ldots, k$
     - Find $e_i = \arg\max_{e \in V \setminus S_{i-1}} \Delta(e|S_{i-1})$ (i.e, the element providing the largest gain)
     - Set $S_i = S_{i-1} \cup \{e_i\}$
  3. Return S_k

- **Theorem.** For any monotone submodular function with $f(\emptyset) = 0$, the greedy algorithm as described above (with k iterations) satisfies

$$f(S_k) \geq \left(1 - \frac{1}{e}\right) f(S_k^\star),$$

  where $S_k^* = \arg\max_{S\,:\,|S|=k} f(S)$.

  - The proof of (a more general version of) the theorem is given below.
  - The factor $1 - \frac{1}{e}$ is *tight*; no algorithm using a *polynomial* number of function calls gets closer in general. (Proved by Nemhauser and Wolsey in 1978.)

- **An important question:** Is getting within $1 - 1/e \approx 0.63$ of the maximum "good enough"?

  - The answer: *It depends.* For a problem like coverage, if the optimal coverage is 100% and an algorithm only guarantees 63% coverage, it seems quite disappointing. But in an influence maximization problem, if influencing a million users would be optimal and we "only" manage to influence around 630,000, that 'eems pretty good.
  - In situations like the former, the more general theorem described below is useful.
  - Note also that 63% is only a worst-case guarantee, and the greedy algorithm often performs much better than the worst-case scenario in practice.

- We can in fact generalize this theorem by letting the size of the set returned (say ℓ) differs from the size of the set whose performance we are comparing against (say k):

- **Theorem.** For any monotone submodular function with $f(\emptyset) = 0$, the greedy algorithm modified to perform ℓ iterations (rather than k iterations) satisfies

$$f(S_\ell) \geq (1 - e^{-\ell/k}) f(S_k^\star)$$

For example, with $\ell = 5k$ iterations we are at least 99.3%-close to $f(S_k^\star)$ (so the set formed has size $5k$, but we are only guaranteeing its performance relative to the best size-k solution).

  - Proof outline (formalized below): Consider the "optimality gap" $f(S_k^*) - f(S_{\text{current}})$. By the greedy selection rule and submodularity, each item closes at least a $\frac{1}{k}$-fraction of gap. Hence, after ℓ iterations, the fraction is at most $\left(1 - \frac{1}{k}\right)^\ell \leq e^{-\ell/k}$.

- **(**Optional**)** There is also a well-known related result for the case that f is *integer-valued*, monotone, and $f(\emptyset) = 0$: To attain a desired target value q (often chosen as $q = f(V)$, the value of the entire set), the greedy algorithm uses at most a $1 + \log(\max_{v \in V} f(\{v\}))$ factor more function evaluations than the smallest possible. (See the early paper "An analysis of the greedy algorithm for the submodular set covering problem" for the case $q = f(V)$.)

  - Here the goal is flipped – instead of "fix the number of evaluations and ask how high a value we can get", the problem is "fix the target value and ask how many evaluations we need to get there".

3.4 Proof Details

We prove the more general theorem. We abbreviate the optimal size-k solution, S_k^*, to simply S^*, and we denote its elements as $\{e_1^\star, e_2^\star, \cdots, e_k^\star\}$. Then for all iterations $i < \ell$, we have

$$
\begin{aligned}
f(S^\star) &\leq f(S^\star \cup S_i) && \text{by monotonicity} \\
&= f(S_i) + \sum_{j=1}^{k} \Delta\left(e_j^\star | S_i \cup \{e_1^\star, e_2^\star, \cdots, e_{j-1}^\star\}\right) && \text{by telescoping} \\
&\leq f(S_i) + \sum_{j=1}^{k} \Delta\left(e_j^\star | S_i\right) && \text{by submodularity} \\
&\leq f(S_i) + \sum_{j=1}^{k} \Delta\left(e_{i+1} | S_i\right) && \text{by def. of greedy selection rule} \\
&\leq f(S_i) + k\left(f(S_{i+1}) - f(S_i)\right). && \text{by } \sum_{j=1}^{k} c = kc \text{ and definition of } \Delta.
\end{aligned}
$$

Re-arranging gives $f(S^*) - f(S_{i+1}) \leq \left(1 - \frac{1}{k}\right)\left(f(S^*) - f(S_i)\right), \forall i < \ell$, and applying this recursively gives

$$
\begin{aligned}
f(S^\star) - f(S_\ell) &\leq \left(1 - \frac{1}{k}\right)^\ell f(S^\star) && \text{since } f(\emptyset) = 0 \\
&\leq e^{-\ell/k} f(S^*). && \text{since } 1 - x \leq e^{-x}, \forall x \in \mathbb{R}
\end{aligned}
$$

Rearranging terms yields $f(S_\ell) \geq (1 - e^{-\ell/k}) f(S^*)$.

4 Other Submodular Maximization Algorithms

The computational complexity of a standard implementation of the greedy algorithm is $O(nk)$, since in each of the k iterations a search is done over $O(n)$ items. This isn't particularly high complexity, but it could

be desirable to reduce it in applications where n is large (e.g., image summarization on millions of images). The following subsections present two variations of the greedy algorithm that seek to address this.

4.1 Lazy Greedy

In an early paper (Minoux, 1978), it was observed that the number of function evaluations done by the greedy algorithm can be reduced (and in turn its computational complexity) without changing the algorithm's output. The idea is outlined as follows, and the algorithm is called Lazy Greedy:

- For each element $e \in V$, keep an upper bound $\rho(e)$ (initialized to ∞) on the marginal gain of adding e. The list of $(e, \rho(e))$ pairs is maintained and kept in decreasing order of $\rho(e)$.
- In a given iteration (say the i-th), evaluate the first element, say e (with the highest $\rho(e)$), and update $\rho(e) \leftarrow \Delta(e|S_{i-1})$. Let e' be the element just after e in the list, and consider the following two cases:
  - If the updated $\rho(e)$ still satisfies $\rho(e) \geq \rho(e')$, then due to the list being sorted, we can conclude that it is the largest in the *entire* list. Hence, the greedy algorithm chooses e, it is removed from further consideration, and we proceed to iteration $i + 1$.
  - If the updated $\rho(e)$ instead satisfies $\rho(e) < \rho(e')$, then we insert e to the suitable later position in the list (to maintain the "decreasing order" property). After doing so, e' is at the start of the list, and we proceed back to the "...evaluate the first element..." step, repeating this process as needed until an element is selected.

Notice that in the first sub-case of Step (ii), although the later items in the list do not get updated, we still maintain the crucial property that $\rho(\cdot)$ is a valid upper bound. This is because, by submodularity, *any updates could only cause $\rho(\cdot)$ to get smaller*, but never larger.

It is difficult to theoretically quantify the precise computational savings of Lazy Greedy, but in practice it can run 10s, 100s, or even 1000s of times faster depending on the problem.

4.2 Stochastic Greedy

The Lazy Greedy algorithm is simply a more efficient way of implementing the greedy algorithm, giving the same answer. A randomized algorithm called Stochastic Greedy is a genuinely different algorithm that maintains the $1 - 1/e$ approximation guarantee but with much lower computation.

The modification is simple: At round i where S_{i-1} has already been chosen, choose N elements of $V \setminus S_{i-1}$ uniformly at random, and among those N, choose the one with the highest $\Delta(e|S_{i-1})$. (The greedy algorithm is similar but considers the entire set $V \setminus S_{i-1}$.)

The key idea is that the analysis for the greedy algorithm shows an improvement in iteration i of $(f(S^*) - f(S_{i-1}))/k$, and this improvement can be obtained by adding a uniformly random element of S^* to the current set (this can be shown using submodularity). Choosing $N = (n/k)\log(1/\epsilon)$ ensures overlapping with S^* with probability at least $1 - \epsilon$, which turns out to be sufficient. (This intuition is a bit imprecise because some elements of S^* might already be in S_{i-1}, but it gives the rough idea.)

Formalizing this idea (see the paper "Lazier than Lazy Greedy") gives the following guarantee: For any non-negative monotone submodular f, the Stochastic Greedy algorithm with $N = (n/k)\log(1/\epsilon)$ guarantees that the final set S_k satisfies

$$\mathbb{E}[f(S_k)] \geq (1 - 1/e - \epsilon) f(S_k^*),$$

where an expectation is included on the left-hand side in view of this being a randomized algorithm.

The runtime is now $O(k \cdot N) = O\big(n \log \frac{1}{\epsilon}\big)$, which is significantly smaller than $O(nk)$ if k is large.

4.3 (**Optional**) An Entirely Different Approach: Multilinear Extension

- Along similar lines to the above notes on submodular minimization, there are techniques for submodular maximizaiton (possibly constrained) based on extending the function from $\{0,1\}^n$ to $[0,1]^n$, applying a continuous optimization algorithm, and then converting back to $\{0,1\}^n$.
- For maximization, a different extension to $[0,1]^n$ turns out to be more useful (rather than the Lovász extension described above), called the *multilinear extension*.
- Again writing $f(S)$ and $f(\mathbf{x})$ interchangeably (where $\mathbf{x}$ has 1s at the entries indexed by S and 0s elsewhere), the extension from $\{0,1\}^n$ to $[0,1]^n$ is as follows:

  $$f(\mathbf{x}) = \mathbb{E}[f(S)]$$

  where element i is included in S with probability x_i, with independence across i values.
- This function is not concave (which would be ideal for maximization), but it does satisfy $\frac{\partial^2 f}{\partial x_i \partial x_j} \leq 0$ for all (i,j), which ensures concavity along lines whose direction vector has all non-negative coefficients.
- Evaluating $f(\mathbf{x})$ exactly is hard, since there are exponentially many possible values of the random S. To circumvent this, one can take random samples from the distribution on S and calculate the average $f(\cdot)$ value with respect to those samples. Then standard concentration bounds (e.g., Hoeffding's inequality) can be used to show that the resulting estimate is accurate with high probability.

4.4 (**Optional**) Beyond Cardinality Constraints

The cardinality constraint $\{S : |S| \leq k\}$ is very common, but certainly not the only kind of constraint we might encounter when optimizing submodular functions. A significantly more general class if constraints is *matroid constraints*. Briefly, given some ground set V and a collection of subsets $\mathcal{I}$ (with each $S \in \mathcal{I}$ being a subset of V), we say that $(V, \mathcal{I})$ is a matroid if:

(i) If $S \in \mathcal{I}$ then any $S' \subset S$ is also in $\mathcal{I}$ (including $S' = \emptyset$);

(ii) If $A \in \mathcal{I}$, $B \in \mathcal{I}$, and $|B| > |A|$, then there exists $x \in B$ such that $A \cup \{x\} \in \mathcal{I}$.

In other words, (i) any removal of elements maintains feasibililty, and (ii) if one feasible set is larger than another, then we can move some element from the latter to the former while maintaining feasibility.

Some examples of matroids are as follows:

- The cardinality-constrained subset that we already studied is a matroid.
- V contains vectors in $\mathbb{R}^d$, and $\mathcal{I}$ consists of all linearly independent subsets.
- $G = (V, E)$ is a graph, and $\mathcal{I}$ contains all the subsets of edges that form a forest.

Returning to submodular maximization, we can now consider the problem

$$\max_{S \in \mathcal{I}} f(S), \tag{2}$$

where f is monotone, submodular, and $f(\emptyset) = 0$, and $(V, \mathcal{I})$ forms a matroid. Then, the standard greedy algorithm provides a $\frac{1}{2}$-approximation, and a more sophisticated algorithm admits a $\left(1 - \frac{1}{e}\right)$-approximation (e.g., see https://chekuri.cs.illinois.edu/papers/submod_max.pdf).

5 (**Optional**) Other Aspects of Discrete Optimization

This section gives a very brief overview of some important aspects of discrete optimization. For further detail, see my MA4254 lecture notes: https://www.comp.nus.edu.sg/~scarlett/MA4254_LectureNotes.pdf

- In the CS department, you can also consider **CS4234 Optimisation Algorithms**

5.1 Integer Linear Programming

Many problems in computer science and applied domains can be cast as an *integer linear program* (ILP):

$$\begin{aligned} &\text{maximize}_{\mathbf{x} \in \mathbb{Z}^d} && \mathbf{c}^T \mathbf{x} && (3) \\ &\text{subject to} && \mathbf{A}\mathbf{x} = \mathbf{b},\ \mathbf{x} \geq \mathbf{0} && (4) \end{aligned}$$

for some matrix $\mathbf{A} \in \mathbb{R}^{m \times d}$ and vectors $\mathbf{b} \in \mathbb{R}^m$ and $\mathbf{c} \in \mathbb{R}^d$, where inequalities between vectors (e.g., $\mathbf{x} \geq \mathbf{0}$) are taken element-wise (e.g., $x_i \geq 0$ for all $i = 1, \ldots, d$).

- The above form is called the *standard form*, and other forms can be converted to this form using some simple tricks. Another common form is that in which we have $\mathbf{A}\mathbf{x} \leq \mathbf{b}$ instead of $\mathbf{A}\mathbf{x} = \mathbf{b}$.

It is especially common for $\mathbf{x}$ to be binary-valued, in which case the constraint $\mathbf{x} \in \{0,1\}^d\}$ (rather than $\mathbb{Z}^d$) may explicitly be included. Having linear constraints and a linear objective function may seem restrictive, but these are in fact very common (e.g., the cardinality constraint that we studied in this lecture is $\sum_{i=1}^n x_i \leq k$, which is linear).

Some examples of problems that can be cast as ILPs are as follows: Knapsack problem, matching problems, covering and packing problems, constraint satisfaction, facilities location problems, scheduling problems, traveling salesman problem (albeit with exponentially many constraints), etc.

5.2 LP Relaxations and Rounding

Solving ILPs is NP-hard in general, and a common approach is to try to relax the integer constraint $\mathbf{x} \in \mathbb{Z}^d$ to $\mathbf{x} \in \mathbb{R}^d$ to produce a *linear program* (LP):

$$\begin{aligned} &\text{maximize}_{\mathbf{x} \in \mathbb{R}^d} && \mathbf{c}^T \mathbf{x} && (5) \\ &\text{subject to} && \mathbf{A}\mathbf{x} = \mathbf{b},\ \mathbf{x} \geq \mathbf{0} && (6) \end{aligned}$$

Similarly, in the binary case, the relaxation is from $\mathbf{x} \in \{0,1\}^d$ to $\mathbf{x} \in [0,1]^d$. Naturally, solving the LP will often give a solution with non-integer values, meaning some sort of *rounding procedure* is needed to produce

something that is feasible for the ILP. This must be done with care, since naive rounding procedures may produce points that, while integer-valued, fail to satisfy $\mathbf{Ax} = \mathbf{b}$ (or $\mathbf{Ax} \leq \mathbf{b}$ in the inequality case).

We briefly summarize the types of guarantees (if any) that might arise using an LP relaxation:

- *(Tight/exact)* If the LP solutions happens to already be integer-valued, then it is also optimal for the ILP. This may sound unrealistic, but it is guaranteed when $\mathbf{A}$ satisfies a property called *totally unimodular*, with examples including matching problems, minimum cut, and shortest path.
- *(Approximate)* After rounding, we might be able to guarantee that the solution is within some factor $\alpha \in (0, 1)$ of optimal (for maximization problems), or is at most an $\alpha > 1$ factor higher than optimal (for minimization problems). This kind of guarantee is common in coverage problems (which are of the minimization type):
  - Vertex cover admits a 2-approximation.
  - More generally, set cover with every element being in at most K sets admits a K-approximation.
  - Using a randomized rounding strategy, the general set cover problem admits an $O(\log n)$-approximation when there are n elements to be covered.
- *(Neither)* In some problems, even getting a constant-factor approximation is NP-hard. For example, solving the Traveling Salesman Problem to within a constant factor would imply solving the Hamiltonian Cycle problem, which is known to be NP-hard.

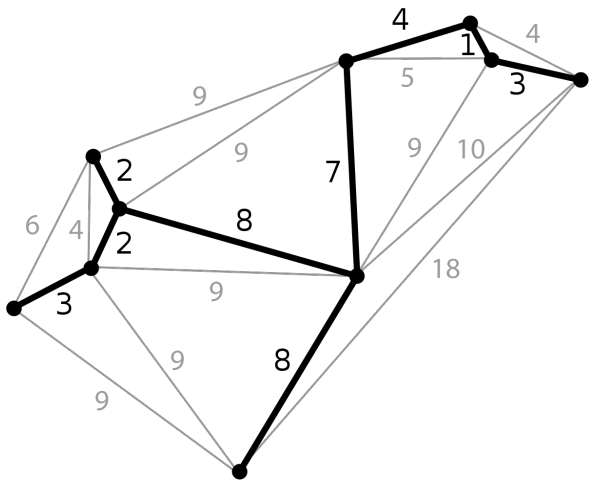
5.3 Greedy Algorithms

In Section 4.4, we discussed submodular maximization subject to a matroid constraint. It turns out that if the function is *modular* (i.e., linear), then we can say something much stronger, namely, a greedy algorithm gives the optimal solution. Concretely, the relevant class of optimization problems is

$$\max_{S \in \mathcal{I}} \sum_{i \in S} c_i, \quad \text{(or min instead of max)}$$

where $(V, \mathcal{I})$ forms a matroid, and $\{c_i\}_{i \in V}$ are fixed constants. Instead of giving a general statement, we state a well-known example of constructing a *minimum spanning tree*, which can be cast as above using the forest matroid that we mentioned earlier.

Let $G = (V, E)$ be an undirected graph with edge weights c_{ij}. A *tree* is a sub-graph for which every two nodes are connected by exactly one path (and hence there are no cycles). The *Minimum Spanning Tree* (MST) problem concerns selecting a subset of edges to form a spanning tree of minimum weight. An example is shown as follows:



The greedy algorithm is simply as follows: *Starting with the empty graph, repeatedly add the smallest-weight edge that does not form a cycle. Stop once there are* $|V| - 1$ *edges.* (Note: Every tree on n nodes has $n - 1$ edges.)

5.4 Global Optimization Methods

There are a number of global methods that guarantee finding the optimal solution for ILPs (and other problems beyond ILPs). While their runtime is exponential in the worst case, they can be very fast in practice. Briefly, some such algorithms include:

- *Branch and bound*, which is a divide-and-conquer approach that efficiently computes upper and lower bounds for various sub-problems (e.g., obtained by setting certain variables to 0 or 1), and using those bounds to rule out large parts of the search space.
- *Cutting plane methods*, which iteratively solve LP relaxations but then introduce further constraints that must hold in the ILP solution, while making the current LP relaxation solution infeasible.
- *Simulated annealing*, which iteratively performs updates with the goal of improving over time while avoiding local optima. Initially ("high temperature") most updates are accepted by the algorithm, but then over time ("cooling down") updates that provide improvements become more preferred, and eventually ("low temperature") only updates that provide improvements are accepted.

Appendix: Proving Submodularity

Exercise 1: *Let* $c_1, \ldots, c_n$ *be fixed non-negative numbers, and consider* $S \subseteq \{1, \ldots, n\}$*. Show that the function with* $f(\emptyset) = 0$ *and* $f(S) = \max_{i \in S} c_i$ *(when* $S \neq \emptyset$*) is submodular and non-decreasing.*

- Solution: The non-decreasing property is immediate from the fact that adding elements to S means taking the maximum over a larger set. To show submodularity, we consider $S \subseteq T$ and consider two cases:
  - If adding j to T doesn't increase the function for T, then we just use the fact that the increase is non-negative for S, and hence $\Delta(j|S) \geq \Delta(j|T)$ as required.

– If adding j to T does increase the function, then it must be because j is the new maximum within $T \cup \{j\}$, and thus also the new maximum within $S \cup \{j\}$. Then by the non-decreasing property, the amount of increase is higher for S than for T.

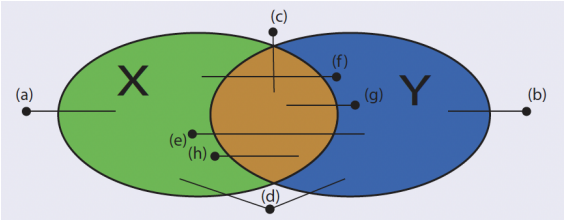
You could also answer this question by comparing $f(A \cup B) + f(A \cap B)$ to $f(A) + f(B)$.

Exercise 2: *Given a graph $G = (V, E)$ and $S \subseteq V$, the cut function $f(S)$ is defined as the number of edges starting in S and ending in $V \setminus S$. Use a diagram to argue that*

$$f(X) + f(Y) \geq f(X \cup Y) + f(X \cap Y)$$

which means that f is submodular. Furthermore, explain why f is non-monotone for any non-empty graph.

- Solution: Consider the categories of edges from the following figure:



Let d_X and d_Y respectively be the contribution to (d) from X and from Y. Then, we have:

– $f(X)$ is the sum of a, c, f, g, d_X
– $f(Y)$ is the sum of b, c, e, h, d_Y
– $f(X \cup Y)$ is the sum of a, b, c, d
– $f(X \cap Y)$ is the sum of c, g, h

Hence, $f(X) + f(Y)$ is $a + b + 2c + d + e + f + g + h$, and $f(X \cup Y) + f(X \cap Y) = a + b + 2c + d + g + h$, so the difference between the two is $e + f \geq 0$.

The function is non-monotone because $f(\emptyset) = f(V) = 0$, but as long as the graph is non empty there exists some S with $\emptyset \subset S \subset V$ such that $f(S) > 0$. So we add nodes starting from the empty set and ending with the full set, at some point the function increases and then decreases again.

CS5275 Lecture 6: Multiplicative Weight Update Algorithms

Jonathan Scarlett

February 22, 2025

Acknowledgment. The first version of these notes was prepared by Chen Jiangwei and Wang Jingtan for a CS6235 assignment.

Useful references:

- Blog post by Jeremy Kun[1]
- A survery on MWU and its applications[2]
- USyd lecture notes: https://ccanonne.github.io/files/compx270-chap12.pdf
- For more advanced material, the book "Prediction, Learning, and Games"

Categorization of material:

- Everything not marked as "Optional" is "Core", except the doubling trick which is "Extra".

(Exam will strongly focus on "Core". Take-home assessments may occasionally require consulting "Extra".)

1 Introduction

In problems such as optimization and prediction, one encounters techniques like gradient descent for *iteratively improving* some candidate solution until it hopefully ends up with a "nearly correct" or "nearly optimal" solution. In particular, gradient descent is based on *additive updates*, which are particularly natural for optimizing real-valued vectors.

Although seemingly less natural at first, similar ideas based *multiplicative updates* can be extremely powerful.[3] This idea has been discovered and re-discovered several times in a diverse variety of research areas, such as machine learning, theoretical computer science, statistics, game theory, and optimization. It is now a major fundamental building block for both theory and practice in these areas; we will discuss some specific applications near the end of the lecture.

[1] https://jeremykun.com/2017/02/27/the-reasonable-effectiveness-of-the-multiplicative-weights-update-algorithm/
[2] https://theoryofcomputing.org/articles/v008a006/
[3] In fact, multiplicative updates and gradient descent aren't as distinct as they may initially seem – a technique known as *mirror descent* unifies and generalizes them both (but this is beyond our scope).

1.1 Problem Setting

In this lecture, we will focus on the problem of *prediction with expert advise*, described as follows:

- There are n **experts**: $i \in \{1, 2, ..., n\}$
- There are T **iterations**: $t \in \{1, 2, ..., T\}$
- In each iteration:
  - Each expert provides some "advice" (to be formalized later). For example, in decision-making problems the advice might be "take the following action: [...]" or in prediction problems the advice might be "predict the following value: [...]".
  - The player decides on a piece of advice to follow.
  - Based on which advice was followed, the player receives a **payoff/reward** (for maximization problems) or a **cost/loss** (for minimization problems). In addition, after making the decision, the player receives knowledge of what the rewards or costs *would have been* for all of the experts (including those not selected).
- Goal: We want to *maximize the summation of rewards* or *minimize a summation of losses* over all iterations.
  - Note that there is no fundamental difference between maximization and minimization – we can just define loss as the negative reward ($\ell = -r$), or if we want to keep both in the range $[0, 1]$ (see below) then $\ell = 1 - r$.
  - **Important:** In this topic, we place *no statistical assumptions at all* on the sequences of rewards/losses. Surprisingly, these sequences can be completely arbitrary and we'll still be able to make strong theoretical statements about learning a good strategy.

As a specific example, let us consider the application of stock market predictions. We aim to predict whether the price of one specific stock will increase or decrease each day.

- We have n *experts*, in T *days*.
- The price of the stock each day either increases or decreases.
- Each day, each expert provides advice: Stock increase or stock decrease (for the specific single stock we are considering)
- The player makes a prediction after seeing all the experts' advice.
- The correct answer is revealed after the current day's prediction is made.
- **Goal:** Minimize the number of wrong predictions (maximize the number of correct predictions.)

An *informal description* of the Multiplicative Weights Update (MWU) algorithm is as follows:

- Initialize n **experts** with identical **weights** w_i for each expert i.
- Update experts' weights multiplicatively and iteratively based on how good their advice is:

  - For each expert i in day t, there is a **loss** $m_i^{(t)}$ based on the expert's advice. (e.g., in the above example, we could set $m_i^{(t)} = 0$ for a correct prediction, and $m_i^{(t)} = 1$ for an incorrect one)
  - Reduce the weights of any experts that performed badly (i.e., made bad advice).

- In each iteration, we decide which experts' advice to follow based on the weights.
- **Goal: Achieve a loss comparable to the best expert's performance.**

Regarding the goal, one may be tempted to aim higher – if there are (say) two experts such that one is correct at certain times and the other is correct at certain different times, couldn't we try to combine the "best of both" and be even better than either one individually? While that may sometimes be possible, it is too much to ask in general. For example, if one expert always predicts "this stock's price will increase" and the other expert always predicts "this stock's price will decrease", then trivially one of them will always be correct, but since neither of them is doing anything "intelligent", we can't expect to achieve any reasonable notion of the "best of both". The notion of competing with the best single expert turns out to provide the right balance between being powerful/desirable but also realistic/attainable.

1.2 Warm-up: When a Perfect Expert Exists

We start with a warm-up problem that is limited in scope but relatively easy to solve:

- Suppose that the predictions are binary, i.e., each expert's advice is a predicted value of 0 or 1.
- Assume that there exists a perfect expert, i.e., one that never makes mistakes.

In the setup, we consider the following algorithm:

- Let $S_t \subseteq \{1, \ldots, n\}$ be the set of experts that don't make a mistake in the first $t-1$ iterations.
- In each iteration t, the algorithm makes a binary prediction, either 0 or 1, by taking a *majority vote over the experts in* S_t. This means that we can view experts not in S_t (i.e., those that have made a mistake) as having been *eliminated*.

Claim: The preceding algorithm makes at most $\log n$ mistakes.
(Note: In this lecture, log *means* $\log_2$*, and* ln *means* $\log_e$*.)*
Proof:

- Initially: $|S_1| = n$
- If a mistake is made in iteration t, then $|S_{t+1}| \leq \frac{|S_t|}{2}$; this is because we took a majority vote over S_t, so a mistake means at least half of them were wrong (so they get removed).
- If m mistakes are made in the first T iterations: $|S_{T+1}| \leq \frac{|S_1|}{2^m} = \frac{n}{2^m}$
- Now we use the assumption that there is a perfect expert. By construction, it will never get eliminated, and hence:

$$\begin{aligned} & \frac{n}{2^m} \geq |S_{T+1}| \geq 1 \\ \implies & n \geq 2^m \\ \implies & \log n \geq m. \end{aligned}$$

1.3 Weighted Majority Algorithm

We now present an algorithm that drops the assumption of a perfect expert existing. This is done by using a weight function to capture each expert's past performance, giving the Weighted Majority Algorithm. We again focus on binary predictions (rather than general rewards/losses), and this time we explicitly introduce the notation for such a setup:

- There is an unknown sequence of correct decisions, $d_*^{(1)}, \ldots, d_*^{(T)}$, each in $\{0, 1\}$;
- At time t, each expert i recommends a decision $d_i^{(t)} \in \{0, 1\}$;
- The correct experts in that round are the ones for which $d_i^{(t)} = d_*^{(t)}$.

Algorithm 1 Weighted Majority Algorithm

1: Fix $0 < \eta \leq \frac{1}{2}$.
2: Initialize the weight $w_i^{(1)} = 1$ for each expert $i \in \{1, \ldots, n\}$.
3: **for** $1 \leq t \leq T$ **do**
4: We make a binary prediction $d^{(t)}$, either 0 or 1, by taking a weighted majority vote: if $\sum_{i:d_i^{(t)}=0} w_i^{(t)} \geq \sum_{i:d_i^{(t)}=1} w_i^{(t)}$ then $d^{(t)} \leftarrow 0$; else $d^{(t)} \leftarrow 1$.
5: Update the weights:
- For every expert that predicted wrongly, set $w_i^{(t+1)} = (1 - \eta) w_i^{(t)}$;
- For every expert that predicted correctly, set $w_i^{(t+1)} = w_i^{(t)}$.

We consider the algorithm described in Algorithm 1. Observe that Line 5 is taking a *weighted majority vote*, and Line 7 is down-weighing the experts that made a wrong prediction. The following theorem gives a guarantee on the number of mistakes made.

Theorem 1.1. *Consider the Weighted Majority Algorithm. Let M_i denote the total number of mistakes made by expert i, and let M denote the total number of mistakes made by the algorithm. Then, for any expert i,*

$$M \leq 2(1 + \eta) M_i + \frac{2 \ln n}{\eta}.$$

To interpret this result, it is useful to think of η as a small constant (e.g, 0.1). If we choose i to index the best expert, then the first term captures having roughly **twice** as many mistakes as the best expert. The second term is only $O(\log n)$ and is thus fairly small unless the number of experts is very large.

Proof. Firstly, we define a *potential function:*

$$\Phi^{(t)} = \sum_{i=1}^{n} w_i^{(t)} \quad \text{(and hence } \Phi^{(1)} = n, \text{ since } w_i^{(1)} = 1 \ \forall i).$$

The idea is to derive both the upper bound and lower bound of the potential function. The bounds will be expressed in terms of M and M_i. Then comparing the two bounds will give the desired inequality involving M and M_i.

To derive an upper bound on $\Phi^{(T+1)}$, consider the case when a mistake is made in iteration t, meaning at least half of the weight is on experts that give wrong advice. Since those experts are the ones down-weighed by $1 - \eta$, it follows that

$$\Phi^{(t+1)} \leq \Phi^{(t)} \Big(\frac{1}{2} + \frac{1}{2}(1 - \eta) \Big) = \Phi^{(t)} \Big(1 - \frac{1}{2} \eta \Big).$$

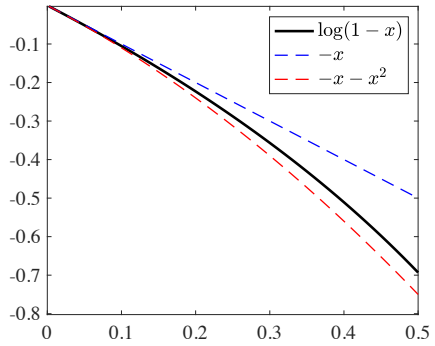


Figure 1: Illustration of bound son $\log(1-x)$.

Stated differently, the first equality holds because half (or more) of the weight decreases by $1-\eta$, and the other half (or less) remains unchanged.

Applying the above inequality inductively, we get

$$\Phi^{(T+1)} \leq \Phi^{(1)}\left(1-\frac{\eta}{2}\right)^M = n \cdot \left(1-\frac{\eta}{2}\right)^M.$$

In addition, we get a trivial lower bound on $\Phi^{(T+1)}$ by just keeping a single term:

$$\Phi^{(T+1)} \geq w_i^{(T+1)} = (1-\eta)^{M_i} w_i^{(1)} = (1-\eta)^{M_i}.$$

Combining the upper and lower bounds, it follows that

$$(1-\eta)^{M_i} \leq n \cdot \left(1-\frac{\eta}{2}\right)^M, \tag{1}$$

or taking the log,

$$M_i \cdot \ln(1-\eta) \leq \ln n + M \cdot \ln\left(1-\frac{\eta}{2}\right).$$

The following inequalities are now useful:

$$-\eta - \eta^2 \leq \ln(1-\eta) \text{ when } 0 < \eta \leq \frac{1}{2} \tag{2}$$

$$\ln\left(1-\frac{\eta}{2}\right) \leq -\frac{\eta}{2} \tag{3}$$

These inequalities can be verified using calculus,[4] but it is easier to see by simply using a picture; see Figure 1. Substituting these inequalities into (1.3), we obtain

$$\begin{aligned} &-M_i \cdot (\eta+\eta^2) \leq M_i \cdot \ln(1-\eta) \leq \ln n + M \cdot \ln\left(1-\frac{\eta}{2}\right) \leq \ln n - M \cdot \frac{\eta}{2} \\ \implies &-M_i \cdot (\eta+\eta^2) \leq \ln n - M \cdot \frac{\eta}{2} \\ \implies &M \leq 2(1+\eta)M_i + \frac{2\ln n}{\eta}. \end{aligned}$$

□

[4] For example, for the first inequality: Let $f(\eta) = \ln(1-\eta)+\eta+\eta^2$, then $f'(\eta) = \frac{1}{\eta-1} + 2\eta + 1 > 0$ when $0 < \eta \leq \frac{1}{2}$. Hence, $f(\eta) \geq f(0) = 0$ when $0 < \eta \leq \frac{1}{2}$. Then, $\ln(1-\eta) \geq -\eta-\eta^2$ when $0 < \eta \leq \frac{1}{2}$.

1.4 Randomized Weighted Majority

The factor of 2 in the above guarantee would ideally be reduced towards 1, but in fact 2 is the best constant possible for deterministic algorithms. For instance, consider a case where there are two experts. Expert 1 always chooses prediction 0 and Expert 2 always chooses prediction 1. If the sequence of correct answers is a "worst-case" one in the sense of making our algorithm perform as poorly as possible, we will make a mistake in all iterations $t \in \{1, \ldots, T\}$. Then, the total number of mistakes made by the algorithm $M = T = M_1 + M_2$, and thus $M \geq 2\min(M_1, M_2)$.

To remove the 2-factor, we may consider randomized algorithms. Intuitively, in examples like the one above, since a fixed strategy might always be wrong, we would actually be better off making completely random decisions, in which case the chance of making mistakes is reduced to 1/2 rather than 1. Accordingly, the randomized weighted majority is similar to the one above, except that it decides each round's prediction using randomization.

Algorithm 2 Randomized Weighted Majority Algorithm

1: Fix $0 < \eta \leq \frac{1}{2}$.
2: Initialize the weight $w_i^{(1)} = 1$ for each expert $i \in \{1, \ldots, n\}$.
3: **for** $1 \leq t \leq T$ **do**
4: We make a binary prediction $d^{(t)}$ randomly, either 0 or 1:
$$\Pr[d^{(t)} = 0] = \frac{\sum_{i:d_i^{(t)}=0} w_i^{(t)}}{\sum_{i=1}^n w_i^{(t)}}; \Pr[d^{(t)} = 1] = \frac{\sum_{i:d_i^{(t)}=1} w_i^{(t)}}{\sum_{i=1}^n w_i^{(t)}}$$
5: Update the weights:
- For every expert that predicted wrongly, set $w_i^{(t+1)} = (1-\eta) w_i^{(t)}$;
- For every expert that predicted correctly, set $w_i^{(t+1)} = w_i^{(t)}$.

See Algorithm 2, particularly Line 5, which is a form of *randomized weighted majority*. As an example, if the weight of experts guessing 0 is the same as the weight of experts guessing 1, then the algorithm is "completely uncertain" so it just makes a 50/50 guess for its prediction (rather than committing to one of the other). In the multiplicative update rule, we down-weigh the experts whose advice was wrong, so that we pay less attention to them in future rounds.

Theorem 1.2. *Consider the Randomized Weighted Majority Algorithm. If we denote M_i as the total number of mistakes made by expert i, and denote M as the total number of mistakes made by the algorithm, for any expert i,*

$$\mathbb{E}[M] \leq (1+\eta) M_i + \frac{\ln n}{\eta}.$$

This guarantee is similar to the previous one, but with $1+\eta$ instead of $2+\eta$, and with an average $\mathbb{E}[\cdot]$ on the left-hand side since the algorithm is now randomized. Thus, by similar reasoning to before, the randomized weighted majority algorithms can achieve approximately the same number of mistakes as the best expert.

Proof. The *potential function* is defined the same as before: $\Phi^{(t)} = \sum_{i=1}^n w_i^{(t)}$ (and hence $\Phi^{(1)} = n$). Similarly to before, we have the lower bound

$$\Phi^{(T+1)} \geq w_i^{(T+1)} = (1-\eta)^{M_i} w_i^{(1)} = (1-\eta)^{M_i}. \tag{4}$$

The main difference in the proof is in getting the upper bound on $\Phi^{(T+1)}$.

Let $d_*^{(t)}$ be the correct label at time t, and let $f^{(t)} = \frac{\sum_{i:d_i^{(t)} \neq d_*^{(t)}} w_i^{(t)}}{\sum_{i=1}^n w_i^{(t)}}$ be the weighted fraction of experts making a mistake in iteration t. Then, the expected number of mistakes made by the algorithm is given by $\mathbb{E}[M] = \sum_{t=1}^T f^{(t)}$ (by Line 6 in the algorithm).

To derive an upper bound of $\Phi^{(T+1)}$, we first express $\Phi^{(t+1)}$ in terms of $\Phi^{(t)}$:

$$\Phi^{(t+1)} = (1 - f^{(t)})\Phi^{(t)} + f^{(t)}(1-\eta)\Phi^{(t)} = \Phi^{(t)}(1 - \eta f^{(t)}) \leq \Phi^{(t)} \cdot e^{-\eta f^{(t)}},$$

where the last step follows from (3). Then, we conduct simple induction to get the upper bound of $\Phi^{(T+1)}$:

$$\Phi^{(T+1)} \leq \Phi^{(1)} e^{-\eta \sum_{t=1}^T f^{(t)}} = n \cdot e^{-\eta \mathbb{E}[M]}$$

Combining this with (4), we have

$$\begin{aligned} (1-\eta)^{M_i} &\leq \Phi^{(T+1)} \leq n \cdot e^{-\eta \mathbb{E}[M]} \\ \implies M_i \ln(1-\eta) &\leq \ln n - \eta \mathbb{E}[M]. \end{aligned}$$

Then, by inequality (2), we have

$$\begin{aligned} -M_i(\eta + \eta^2) \leq M_i \ln(1-\eta) &\leq \ln n - \eta \mathbb{E}[M] \\ \implies \mathbb{E}[M] &\leq (1+\eta) M_i + \frac{\ln n}{\eta}. \end{aligned}$$

□

2 Multiplicative Weights Update (MWU) Algorithm

So far, we have only considered the case of binary prediction, and considered algorithms that multiply the weights by $1-\eta$ whenever a wrong prediction is made. More generally, we might be interested in real-valued prediction, or in settings where choosing an expert incurs some real-valued "loss". (Letting the loss be in $\{0, 1\}$ recovers the earlier setting of "no mistake vs. mistake".)

To handle such scenarios, we consider a general setup in which some real number $m_i^{(t)}$ represents the loss (or "negative reward") when taking the advice of an expert i in iteration t. Handling arbitrarily large losses turns out to be infeasible, so we impose the boundedness assumption $m_i^{(t)} \in [-1, 1]$.

- Note: If the losses are known to be between $[-\rho, \rho]$ for some $\rho > 0$, they can be divided by ρ (i.e., normalized) to become between $[-1, 1]$ (see the tutorial for details), so this assumption is not as restrictive as it may seem.

In more detail, in iteration $t \in \{1, 2, \ldots, T\}$, the following happens:

- The algorithm selects a probability distribution $\boldsymbol{p}^{(t)}$ over the experts: $\boldsymbol{p}^{(t)} = (p_1^{(t)}, \ldots, p_n^{(t)})$, where $p_i^{(t)} = \Pr[\text{expert i's advice is followed at time } t]$. Since this is a probability distribution, we have $\sum_{i=1}^n p_i^{(t)} = 1$
- After $\boldsymbol{p}^{(t)}$ is chosen, the losses $m_i^{(t)}$ are revealed.

- The expected loss for the current iteration t is: $\mathbb{E}_{i\sim \boldsymbol{p}^{(t)}}[m_i^{(t)}] = m^{(t)} \cdot \boldsymbol{p}^{(t)} = m_1^{(t)} p_1^{(t)} + m_2^{(t)} p_2^{(t)} + \ldots + m_n^{(t)} p_n^{(t)}$, where here and subsequently, $\mathbf{u} \cdot \mathbf{v} = \sum_i u_i v_i$ denotes the regular inner proudct (dot product) between two vectors.

As before, we take each $\{m_i^{(t)}\}_{t=1}^T$ to be a completely arbitrary sequence (in particular, these are *not* necessarily produced by a probabilistic process). In fact, we can even handle "adversarial" settings where, for each t, the losses $\{m_i^{(t)}\}_{i=1}^n$ are generated based on the algorithm's choices up to that time, in a manner that may make the problem as challenging as possible.

The MWU algorithm is described in Algorithm 3.

Algorithm 3 Multiplicative Weights Algorithm

Fix $0 < \eta \leq \frac{1}{2}$.
Initialize the weight $w_i^{(1)} = 1$ for each expert $i \in \{1, \ldots, n\}$.
for $1 \leq t \leq T$ **do**
    Let $\Phi^{(t)} = \sum_{i=1}^n w_i^{(t)}$.
    Select the distribution $\boldsymbol{p}^{(t)} = \left(\frac{w_1^{(t)}}{\Phi^{(t)}}, \frac{w_2^{(t)}}{\Phi^{(t)}}, \ldots, \frac{w_n^{(t)}}{\Phi^{(t)}}\right)$.
    Update the weight for every expert: $w_i^{(t+1)} = (1 - \eta m_i^{(t)}) w_i^{(t)}$.

Our objective remains to show that the sum of expected losses (over $t = 1, \ldots, T$) is not too much more than the loss of the best expert. Formally, we have the following.

Theorem 2.1. *With $\sum_{t=1}^T m_i^{(t)}$ being the total loss if we always follow expert i and $\sum_{t=1}^T \boldsymbol{m}^{(t)} \cdot \boldsymbol{p}^{(t)}$ as the expected loss of the algorithm, we have for any expert i that the MWU algorithm satisfies*

$$\sum_{t=1}^T \boldsymbol{m}^{(t)} \cdot \boldsymbol{p}^{(t)} \leq \sum_{t=1}^T m_i^{(t)} + \eta \sum_{t=1}^T |m_i^{(t)}| + \frac{\ln n}{\eta}.$$

Before giving the proof, we discuss the terms and give a simpler weakened version. The left-hand side represents the average loss incurred by the algorithm, where at time step t we average over the chosen distribution $\boldsymbol{p}^{(t)}$. On the right-hand side, if we choose i to be the best expert, then the first term is simply that expert's total loss. The second term comes from online decision making incurring slight suboptimality; for example, if all $m_i^{(t)}$ are positive then it just amounts to having $(1 + \eta)$ times the smallest loss, which is only a minor increase if η is small. The final term again gives a mild logarithmic dependence on n, though due to its presence we need to avoid η being *too* small.

To get a weakened (but simpler) bound, we can use $\sum_{t=1}^T |m_i^{(t)}| \leq T$, so that the increase over the best expert's loss (known as the *regret*) is at most $T\eta + \frac{\ln n}{\eta}$. A simple differentiation exercise shows that this is minimized when $\eta = \sqrt{\frac{\ln n}{T}}$, giving a regret of at most $2\sqrt{T \ln n}$, i.e.,

$$\sum_{t=1}^T \boldsymbol{m}^{(t)} \cdot \boldsymbol{p}^{(t)} \leq \sum_{t=1}^T m_i^{(t)} + 2\sqrt{T \ln n}.$$

Note that $\sqrt{T}$ could be significant, but it is *sub-linear*, whereas the leading term $\sum_{t=1}^T m_i^{(t)}$ often grows linearly. If $\sum_{t=1}^T m_i^{(t)}$ only grows slowly due to most/all of the $m_i^{(t)}$ terms being near zero, then the upper bound in the theorem will tend to be significantly better than the weakened bound.

We now proceed with the proof of the theorem.

Proof. Similar to the previous proof, the goal is to calculate the upper and lower bounds of the potential function (again defined via $\Phi^{(t)} = \sum_{i=1}^{n} w_i^{(t)}$), expressed using the relevant loss terms. By comparing these limits, we can establish the required inequality that includes $\boldsymbol{m}^{(t)}$ and $m_i^{(t)}$.

Upper bound on Φ: We start with an upper bound on the potential function:

$$\begin{aligned}
\Phi^{(t+1)} &= \sum_{i=1}^{n} w_i^{(t+1)} \\
&= \sum_{i=1}^{n} (1 - \eta m_i^{(t)}) w_i^{(t)} \qquad \text{(by the update rule)} \\
&= \sum_{i=1}^{n} (1 - \eta m_i^{(t)}) p_i^{(t)} \Phi^{(t)} \qquad \text{(by the choice of } p_i^{(t)} \text{ in the algorithm)} \\
&= \Phi^{(t)} - \eta \Phi^{(t)} \sum_{i=1}^{n} m_i^{(t)} p_i^{(t)} \qquad \text{(since the } p_i^{(t)} \text{ sum to one)} \\
&= \Phi^{(t)} (1 - \eta \boldsymbol{m}^{(t)} \cdot \boldsymbol{p}^{(t)}) \\
&\leq \Phi^{(t)} e^{-\eta \boldsymbol{m}^{(t)} \cdot \boldsymbol{p}^{(t)}}. \qquad \text{(since } 1 - a \leq e^{-a}\text{)}
\end{aligned}$$

Applying this inequality recursively, we obtain

$$\Phi^{(T+1)} \leq \Phi^{(1)} \cdot e^{-\sum_{t=1}^{T} \eta \boldsymbol{m}^{(t)} \cdot \boldsymbol{p}^{(t)}} = n \cdot e^{-\sum_{t=1}^{T} \eta \boldsymbol{m}^{(t)} \cdot \boldsymbol{p}^{(t)}}. \tag{5}$$

Lower bound on Φ: Since $\Phi^{(T+1)}$ is a sum of non-negative weights, we can easily get a lower bound by taking any single weight (say indexed by i):

$$\Phi^{(T+1)} \geq w_i^{(T+1)} = \prod_{t=1}^{T} (1 - \eta m_i^{(t)}).$$

Next, we utilize the following inequalities, which can fairly easily derived from the convex nature of exponential functions (but we will skip the proofs):

$$(1 - \eta)^x \leq 1 - \eta x \text{ if } x \in [0, 1]$$

$$(1 + \eta)^{-x} \leq 1 - \eta x \text{ if } x \in [-1, 0].$$

Since $m_i^{(t)} \in [-1, 1]$, for every expert i, these inequalities allow us to weaken the lower bound on $\Phi^{(T+1)}$ as follows:

$$\Phi^{(T+1)} \geq (1 - \eta)^{\sum_{i\,:\,m_i^{(t)} \geq 0} m_i^{(t)}} \cdot (1 + \eta)^{-\sum_{i\,:\,m_i^{(t)} < 0} m_i^{(t)}}. \tag{6}$$

(Note: This equation and the rest of the analysis become simpler if we assume that the losses are in $[0, 1]$ *instead of* $[-1, 1]$*, but we will handle the latter (more general) case.)*

Combining the bounds: By (5) and (6):

$$n \cdot e^{-\sum_{t=1}^{T} \eta \boldsymbol{m}^{(t)} \cdot \boldsymbol{p}^{(t)}} \geq (1 - \eta)^{\sum_{t\,:\,m_i^{(t)} \geq 0} m_i^{(t)}} \cdot (1 + \eta)^{-\sum_{t\,:\,m_i^{(t)} < 0} m_i^{(t)}}.$$

Taking the logarithm on both sides, we get

$$\ln n - \sum_{t=1}^{T} \eta \boldsymbol{m}^{(t)} \cdot \boldsymbol{p}^{(t)} \geq \sum_{t\,:\,m_i^{(t)} \geq 0} m_i^{(t)} \ln(1-\eta) - \sum_{t\,:\,m_i^{(t)} < 0} m_i^{(t)} \ln(1+\eta). \tag{7}$$

Then, negating and scaling by $1/\eta$ gives:

$$\begin{aligned}
\sum_{t=1}^{T} \boldsymbol{m}^{(t)} \cdot \boldsymbol{p}^{(t)} &\leq \frac{\ln n}{\eta} - \frac{1}{\eta} \sum_{t\,:\,m_i^{(t)} \geq 0} m_i^{(t)} \ln(1-\eta) + \frac{1}{\eta} \sum_{t\,:\,m_i^{(t)} < 0} m_i^{(t)} \ln(1+\eta) \\
&= \frac{\ln n}{\eta} + \frac{1}{\eta} \sum_{t\,:\,m_i^{(t)} \geq 0} m_i^{(t)} \ln \frac{1}{(1-\eta)} + \frac{1}{\eta} \sum_{t\,:\,m_i^{(t)} < 0} m_i^{(t)} \ln(1+\eta) \\
&\leq \frac{\ln n}{\eta} + \frac{1}{\eta} \sum_{t\,:\,m_i^{(t)} \geq 0} m_i^{(t)} (\eta + \eta^2) + \frac{1}{\eta} \sum_{t\,:\,m_i^{(t)} < 0} m_i^{(t)} (\eta - \eta^2) \\
&= \frac{\ln n}{\eta} + \frac{1}{\eta} \left(\eta \sum_{t\,:\,m_i^{(t)} \geq 0} m_i^{(t)} + \eta \sum_{t\,:\,m_i^{(t)} < 0} m_i^{(t)} + \eta^2 \sum_{t\,:\,m_i^{(t)} \geq 0} m_i^{(t)} - \eta^2 \sum_{t\,:\,m_i^{(t)} < 0} m_i^{(t)} \right) \\
&= \frac{\ln n}{\eta} + \sum_{t=1}^{T} m_i^{(t)} + \eta \sum_{t\,:\,m_i^{(t)} \geq 0} m_i^{(t)} - \eta \sum_{t\,:\,m_i^{(t)} < 0} m_i^{(t)} \\
&= \frac{\ln n}{\eta} + \sum_{t=1}^{T} m_i^{(t)} + \eta \sum_{t=1}^{T} |m_i^{(t)}|,
\end{aligned}$$

where the second inequality is due to:

$$\ln \frac{1}{1-\eta} \leq \eta + \eta^2 \text{ when } \eta \in [0, 1/2]$$
$$\ln(1+\eta) \geq \eta - \eta^2 \text{ when } \eta \in [0, 1/2].$$

(The first of these was shown in Figure 1 (after multiplying -1 on both sides), and the second is analogous.) □

We conclude with some discussion:

- There are two hyperparameters in the MWU algorithm, namely, η and T. The number of iterations T may be dictated by the application, or we may be free to choose it "large enough" to get some desired guarantee (e.g., guarantee a loss at most $1+\epsilon$ higher than that of the best expert for some small ϵ). The parameter η is analogous to the "step size" in other optimization algorithms; it could be chosen according to the weakened bound (as we discussed after the theorem statement), could be set to a fixed value like 0.1, or could be tuned based on data (if available).
- After Theorem 2.1, we derived the choice $\eta = \sqrt{\frac{\ln n}{T}}$, but it should be noted that such a choice relies on knowing T. However, the value of T is not always known in advance. One way to circumvent this issue is to use the *doubling trick* at the price of a small constant factor in the regret bound. The idea is to divide T into intervals of known lengths, then we can set η separately within each interval. It can then be shown that summing up the losses at each interval is not much worse than the optimal one. More details can be found in Sec. 8.2 of the book "Foundations of Machine Learning".

- In the assumption of losses being in $[-1, 1]$, the use of the number 1 is simply for convenience. It is straightforward to extend to $[-\rho, \rho]$ for arbitrary $\rho > 0$, since the learner can always choose to divide the rewards by ρ. The resulting guarantee on the *original* reward changes slightly due to this scaling (see the tutorial for details) but remains fundamnetally similar to the case of $[-1, 1]$-valued losses.

2.1 (**Optional**) The Hedge Algorithm

We briefly mention that there is another algorithm called the Hedge algorithm that closely resembles the MWU algorithm that we studied above. It differs in the following aspects:

- Range of η: The Hedge algorithm is less restrictive by only requiring $0 < \eta \leq 1$.
- Weight update scheme: The Hedge algorithm uses an slightly different exponential update.

See Algorithm 4 for the details.

Algorithm 4 The Hedge Algorithm

Fix $0 < \eta \leq 1$.
Initialize the weight $w_i^{(1)} = 1$ for each expert $i \in \{1, \dots, n\}$.
for $1 \leq t \leq T$ **do**
    Let $\Phi^{(t)} = \sum_{i=1}^{n} w_i^{(t)}$.
    Select the distribution $\boldsymbol{p}^{(t)} = \left(\frac{w_1^{(t)}}{\Phi^{(t)}}, \frac{w_2^{(t)}}{\Phi^{(t)}}, \dots, \frac{w_n^{(t)}}{\Phi^{(t)}} \right)$.
    Update the weight for every expert: $w_i^{(t+1)} = w_i^{(t)} \cdot \exp(-\eta m_i^{(t)})$.

Some notes:

- The main difference is multiplying by $\exp(-\eta m_i^{(t)})$ instead of $1 - \eta m_i^{(t)}$. These two quantities are closely related via $1 - x \leq e^{-x}$, with the two quantities being increasingly similar as x decreases. In view of this inequality, the resulting guarantee for Hedge follows fairly similarly to MWU.
- In slight more detail: (i) The analysis is still based on upper and lower bounding $\Phi^{(T+1)}$; (ii) The lower bound on $\Phi^{(T+1)}$ is simpler due to the exponential update rule; (iii) The upper bound on $\Phi^{(T+1)}$ is a bit more complicated and requires this inequality $e^{-\eta x} \leq 1 - \eta x + \eta^2 x^2$ for $|\eta x| \leq 1$.

Skipping the remaining details, the resulting theorem is stated as follows without proof.

Theorem 2.1. *Assuming that all losses satisfy $m_i^{(t)} \in [-1, 1]$ and $\eta \in [0, 1]$, the Hedge algorithm guarantees that after T rounds, for any expert $i \in \{1, \dots, n\}$,*

$$\sum_{t=1}^{T} \boldsymbol{m}^{(t)} \cdot \boldsymbol{p}^{(t)} \leq \sum_{t=1}^{T} m_i^{(t)} + \eta \sum_{t=1}^{T} \left(\boldsymbol{m}^{(t)} \right)^2 \cdot \boldsymbol{p}^{(t)} + \frac{\ln n}{\eta}$$

where $\left(\boldsymbol{m}^{(t)} \right)^2$ is the coordinate-wise square of $\boldsymbol{m}^{(t)}$.

Observe that the right-hand side above depends on $\boldsymbol{p}^{(t)}$, while the guarantee of MWU only depends on the loss of the best expert in hindsight, which gives stronger bounds in some applications. For example, in approximation of many NP-hard problems, we explicitly require the use of MWU to obtain better bounds than the Hedge algorithm.

On the other hand, we still have $\sum_{t=1}^{T} \left(\boldsymbol{m}^{(t)} \right)^2 \cdot \boldsymbol{p}^{(t)} \leq T$, and if we weaken the second term in this way, we end up with exactly the same weakened bound as MWU, with a regret of $2\sqrt{T \ln n}$.

3 (**Optional**) Beyond Bounded Losses

We have focused on losses $m_i^{(t)}$ that lie in $[-, 1]$, but are otherwise arbitrary. This means that we are very flexible in being able to handle many different sequences of losses (other than the boundedness restriction).

On the other hand, there are certain specific loss functions where we can get significantly stronger guarantees than the above theorems, or where we can allow for unbounded loss values under which the above theorems are not even applicable. Two such losses are as follows:

- **Logarithmic loss:** Suppose a sequence $x_1, \ldots, x_T$ is given sequentially to the learner, who at time t (having seen $x_1, \ldots, x_{t-1}$), is required to choose[5] some $\theta_t \in \Theta$, where each θ is a parameter (or vector of parameters) associated with a probability density function $f_\theta(x)$. For example, $\theta = (\mu, \sigma^2)$ could represent the mean/variance parameters of a Gaussian distribution.

  We are interested in finding the density function that "best models" $x_1, \ldots, x_T$. For doing so, a natural choice is the *logarithmic loss* (since it's related to maximum-likelihood estimation; see below):

  $$\ell_\theta(x) = \log \frac{1}{f_\theta(x)}.$$

  This leads to the following notion of "regret" (total loss incurred compared to the smallest possible):

  $$R_T = \sum_{t=1}^{T} \log \frac{1}{f_{\theta_t}(x_t)} - \min_{\theta \in \Theta} \sum_{t=1}^{t} \log \frac{1}{f_\theta(x_t)}.$$

  Note that $\sum_{t=1}^{t} \ell_\theta(x_t) = -\log\big(\prod_{t=1}^{T} f_\theta(x_t)\big)$, so what we are essentially doing is seeking θ that maximizes the likelihood of the data, thus the connection to maximum-likelihood estimation.

  Like in the bounded losses setting, it turns out to be beneficial to let the algorithm randomize its decision and return a *distribution w_t over θ values*, leading to the following generalization:

  $$R_T = \sum_{t=1}^{T} \log \frac{1}{\mathbb{E}_{\theta \sim w_t}[f_\theta(x_t)]} - \min_{\theta \in \Theta} \sum_{t=1}^{t} \log \frac{1}{f_\theta(x_t)}.$$

- **Squared loss:** Suppose that instead of just an x-sequence, a sequence of (x_t, y_t) pairs is given, and at time t the learner is required to predict y_t given both x_t and all the previous pairs. This is a *regression* problem if y is real-valued, and a *classification problem* is y is binary-valued. In the former case, a natural loss is the square loss:

  $$\ell(y, \hat{y}) = (y - \hat{y})^2.$$

  This loss is also well-suited to MWU algorithms, albeit not quite to the extent of the logarithmic loss. We will not consider it further here.

In the following, we focus on the logarithmic loss, or log loss for short. The MWU algorithm maintains a set of weights over all the θ values, taking the form

$$w_t(\theta) \propto \exp\Big(-\eta \sum_{t=1}^{t-1} \ell_\theta(x_i)\Big)\pi(\theta),$$

[5] If Θ is a finite set then we can interpret each $\theta \in \Theta$ as an "expert", but here we could also have a continuous set.

where $\eta > 0$ is a parameter (typically 1 when log loss is used) and $\pi(\theta)$ is a "prior" (often taken to be uniform if Θ is a finite or compact set). The symbol $\propto$ means the right-hand side should be scaled so that the weights sum (or integrate) to one.

We observe that the above formula closely resembles what we used in the case of bounded losses, and we can similarly interpret it as a multiplicative update: After observing x_t, we update

$$w_{t+1}(\theta) \propto w_t(\theta) e^{-\eta \ell_\theta(x_i)},$$

noting that the multiplication becomes summation inside the exponential.

In the case that Θ is a finite set of size n and $\pi(\cdot)$ is uniform, it can be shown that $R_T \leq \log n$, which is significantly stronger than what we got for bounded losses – the right-hand side is no longer $O(\sqrt{T})$, but instead has no dependence on T at all! (Though the dependence on n remains similar.) Results for continuous sets Θ are also well-known. For more on this and also the square loss, see (e.g.) the tutorial slides https://maths.anu.edu.au/sites/prod.maths.sca-lws06.anu.edu.au/files/Nikita%20SummerSchoolFinal.pdf.

4 (**Optional**) Applications of MWU

In this section, we survey several applications and uses of multiplicative weights update algorithms, some of which are far from obvious. Even more applications can be found in Section 3 of the survey https://theoryofcomputing.org/articles/v008a006/.

4.1 Machine Learning Algorithms

One of the most fundamental machine learning problems is *binary classification*, where we are given a training set $\{(x_t, y_t)\}_{t=1}^n$ with $x \in \mathbb{R}^d$ and $y_t \in \{-1, 1\}$ (binary labels) and we want to devise a "good" predictor $\hat{y} = f(x)$. A natural starting point is to consider *linear predictors*, taking the form $f_\theta(x) = \langle \theta, x \rangle$, and the task becomes finding a good choice of θ.

One of the earliest algorithms for this task is the *Perceptron algorithm*, which iterates through the training set and updates $\theta \leftarrow \theta + x_t y_t$ whenever a mistake is made. This is an *additive update*. A lesser-known algorithm called the **Winnow algorithm** instead performs *multiplicative updates*, and comes with theoretical guarantees that resemble those of the Perceptron algorithm, but can also be much stronger in some cases. See Section 3.1 of https://theoryofcomputing.org/articles/v008a006/.

Another famous algorithm based on MWU is **AdaBoost**, which roughly operates as follows:

- Maintain a collection of weights (summing to one) over the training points;
- In each iteration, choose a classifier (typically from a "simple" class) that minimizes the weighted training error;
- Using the *multiplicative weights method*, up-weigh the points that are mis-classified by that classifier (so that we give them more attention later and "correct" for these mistakes), and down-weigh the ones that are classified correctly.
- Continue until some stopping criterion, and let the final classifier be a suitably-weighted combination of those constructed along the way (i.e., a form of weighted majority vote).

AdaBoost has a number of interesting theoretical guarantees and is also widely adopted in practice. See https://www.comp.nus.edu.sg/~scarlett/CS5339_notes/ for some lecture notes on this topic, or https://jeremykun.com/2015/05/18/boosting-census/ for an introductory blog post.

4.2 Solving Zero-Sum Games

Consider a setting in which there are two agents that choose values $i \in \{1, \dots, n\}$ and $j \in \{1, \dots, n\}$, from which a value $A(i, j) \in [0, 1]$ is formed that the first player wants to minimize and the second player wants to maximize. More generally, the players could randomize their choice, producing distributions $\mathbf{p}$ and $\mathbf{q}$ and an average value of $A(\mathbf{p}, \mathbf{q}) = \mathbb{E}_{i \sim \mathbf{p}, j \sim \mathbf{q}}[A(i, j)]$.

Once one player's (randomized) strategy is fixed, it is easy to show that the other's best strategy is deterministic. Using this observation along with the minimax theorem for convex/concave functions (note that the quantity $\mathbb{E}_{i \sim \mathbf{p}, j \sim \mathbf{q}}[A(i, j)]$ is linear in $\mathbf{p}$ and linear in $\mathbf{q}$), we find that

$$\min_{\mathbf{p}} \max_{j} A(\mathbf{p}, j) = \max_{\mathbf{q}} \min_{i} A(i, \mathbf{q}) =: A^*.$$

The MWU algorithm can be used to not only prove this result, but also efficiently find a pair $(\mathbf{p}, j)$ or $(\mathbf{q}, i)$ whose min-max or max-min value is arbitrarily close to A^*.

The idea is to let the MWU losses be $m_i^{(t)} = A(i, j^{(t)})$, where $j^{(t)}$ is the best deterministic choice for player 2 given player 1's mixed strategy $\mathbf{p}^{(t)}$ formed after the previous round. The intuition is that the total loss incurred by $\mathbf{p}^{(1)}, \dots, \mathbf{p}^{(T)}$ cannot be much higher than the loss of any fixed choice of i (by the preceding theorems), and thus not much higher than the average loss over $i \sim \mathbf{p}$ for any fixed distribution $\mathbf{p}$. When we choose $\mathbf{p} = \mathbf{p}^*$ with $\mathbf{p}^*$ solving the min-max problem $\min_{\mathbf{p}} \max_j A(\mathbf{p}, j)$, it follows that the loss approaches A^*. See Section 3.2 of https://theoryofcomputing.org/articles/v008a006/ for the details.

4.3 Solving Linear Programs

MWU has been applied to solving covering/packing linear programs where the coefficients are non-negative. The idea is to reduce optimizing a linear program to feasibility testing problems then perform binary search on the feasibility variant. The MWU method comes in in solving the feasibility problem approximately. Each constraint in the LP corresponds to an expert. The loss incurred by an expert is defined to be how much the constraint is violated. The intuition is that by putting more weights on the constraints that are not satisfied, we will eventually violate fewer constraints and find a feasible solution as number of iterations increases. More details can be found in Sec. 3.3 of https://theoryofcomputing.org/articles/v008a006/, or the later part of the blog post https://www.jeremykun.com/2017/02/27/the-reasonable-effectiveness-of-the-multiplicative-weights-update-algorithm/.

4.4 Bandit Algorithms

In this lecture, we assumed that after each decision is made, the loss associated with *all experts* are revealed to the learner. That is, after each round, we know how good or bad each expert's decision would have been had we chosen it. A more challenging setting is that of *bandit feedback*, in which we only get to observe the loss associated with the single expert that we chose. This may be more natural in applications, e.g., imagine each "expert" corresponds to a choice of advertisement to display to a user with the hope that they click on

it – we get to observe whether they did so, but we can only guess as to whether they would have clicked on some other one.

Despite the considerably reduced amount of information available the bandit setting, we can still use the MWU idea. The idea is to replace each loss $m_i^{(t)}$ by a surrogate that is *correct on average*, which can be done by setting unobserved losses to 0 and dividing observed losses by the probability assigned to the expert. This gives rise to the *EXP3 algorithm*, which incurs regret at most $2\sqrt{nT \log n}$ after T rounds with n experts. This means that the regret has a similar dependence on T as the full information setting, but a stronger dependence on n. See Chapter 11 of https://tor-lattimore.com/downloads/book/book.pdf for the details, or https://jeremykun.com/2013/11/08/adversarial-bandits-and-the-exp3-algorithm/ for an introductory blog post.

4.5 Others

Some other applications are briefly mentioned as follows:

- Online convex optimization (see the tutorial for a question on this)
- Noisy binary search (see "Noisy binary search and its applications")
- Learning statistical graphical models (see "Learning graphical models using multiplicative weights")
- Approximating integer programs (see "Randomized rounding without solving the linear program")
- Semidefinite programming (see "Fast algorithms for approximate semidefinite programming using the multiplicative weights update method")
- Differential privacy (see Sec. 11.2 of "The algorithmic foundations of differential privacy")
- Constrained consensus optimization (see "Consensus multiplicative weights update: Learning to learn using projector-based game signatures")

CS5275 Lecture 7: The Fourier Transform

Jonathan Scarlett

March 8, 2025

Acknowledgment. The first version of these notes was prepared by Lau Kang Ruey Gregory and Zhang Yang for a CS6235 assignment.

Useful references:

- 3Blue1Brown: https://www.youtube.com/playlist?list=PL4VT47y1w7A1-T_VIcufa7mCM3XrSA5DD
- Blog post primers – 'Fourier Analysis' under https://www.jeremykun.com/primers/
- Textbook "Principles of Fourier Analysis" by Kenneth B. Howell
- A Very Short Course on Time Series Analysis: https://bookdown.org/rdpeng/timeseriesbook/
- *(Not a topic we'll cover but particularly relevant to TCS)* Fourier analysis for analyzing Boolean functions: https://www.youtube.com/watch?v=lXJP-UkTl-4

Categorization of material:

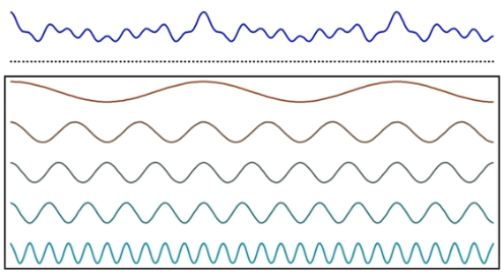
- Core material: Sections 1–4
- Extral material: Section 5 (DFT and FFT)

(Exam will strongly focus on "Core". Take-home assessments may occasionally require consulting "Extra".)

1 Introduction

1.1 Overview

In simple terms, the Fourier transform is a tool that allows us to represent a real-valued function (i.e., a "signal", or an "image" if the coordinates are 2D) as a superposition (sum/integral) of sines and cosines (e.g., $\sin(\omega x)$ for various ω) or complex exponentials (e.g., $e^{i\omega x}$ for various ω) with suitably-chosen amplitudes and frequencies. An example from 3Blue1Brown (the signal at the top is a weighted combination of the signals below):



In some cases, the benefit of such representations is immediate, e.g.:

- If a signal that we are interested has all of its energy a certain frequency range but it gets corrupted by random noise (which typically spans all frequencies), we can immediately mitigate the noise by replacing "noise-only" frequency components by zero.
- In audio applications, we can seamlessly perform operations like "increase the bass level" as well as more complex tasks.
- In telecommunications, different transmitters can transmit their data within different frequency bands to avoid interfering with each other, and the Fourier transform facilitates this.

In other cases, the Fourier transform may be used in a more indirect or abstract manner, e.g.:

- Sinusoids and exponentials frequently arise as solutions to differential equations, and the Fourier transform can be very useful in finding such solutions more easily.
- When computing Ax for some $n \times n$ matrix A and length-n vector x (say), for certain classes of the matrix A (e.g., circulant), the Fourier transform can be used to perform the multiplication in time $O(n \log n)$ instead of $O(n^2)$.
- Fourier transforms are widely used in probability under the name *characteristic functions*.
- A Boolean version of the Fourier transform is widely used in the analysis of Boolean functions.

The practical impact of the Fourier transform is staggering – it is constantly used in our phones, computers, etc., and an algorithm called the 'Fast Fourier Transform (FFT)' may well be one of the most invoked algorithms ever.

1.2 Background on Bases

The Fourier transform can be viewed as a convenient choice of *basis* for (a class of) real-valued functions. Since the notion of basis may be more familiar for vectors rather than functions, it's useful to draw an analogy between the two.

First consider real vectors in Euclidean space, say $\mathbb{R}^d$, with the usual inner product $\langle u, v \rangle = \sum_{i=1}^{n} u_i v_i$, and note the following:

- The *standard basis* is given by $\{e_i\}_{i=1}^{n}$, where $e_i = (0, \ldots, 0, 1, 0, \ldots, 0)$ with the 1 being in the i-th position. The meaning of *basis* is that (i) any $u \in \mathbb{R}^d$ can be written as a linear combination of these, and (ii) none of the basis vectors can be written as a linear combinations of the others.
- Other bases are also possible – for example, in $\mathbb{R}^2$ we could write any vectors as a linear combination of $(0, 1)$ and $(1, 1)$, or a linear combination of $\left(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right)$ and $\left(-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right)$.
- Some bases are more "convenient" than others for certain purposes – for example, for a matrix $A \in \mathbb{R}^{n \times n}$, the eigenvalue decomposition can be viewed as *transforming to a basis in which the operation A becomes coordinate-wise scaling*.

- One particularly useful property a basis $\{v_i\}_{i=1}^n$ can have is *orthonormality*: $\langle v_i, v_j \rangle = 0$ for all i, j (orthogonal) and $\|v_i\| = 1$ for all i (normalized). In this case, we can write any vector $u \in \mathbb{R}^d$ as

$$u = \sum_{i=1}^{n} \langle u, v_i \rangle v_i,$$

  with each term $\langle u, v_i \rangle v_i$ being interpreted as the projection of u onto v_i.

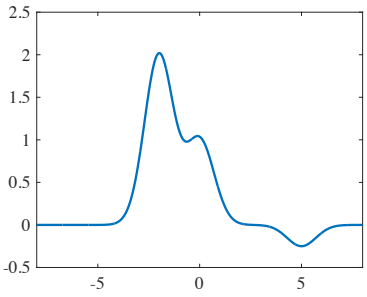
Now we move from vectors to real-valued functions, i.e., $f(x)$ with $x \in \mathbb{R}$. Here x replaces the role of the index $i \in \{1, \ldots, n\}$ compared to the vector case. The above considerations naturally generalize as follows, with summations now replaced by integrals:

- We use the following notion of inner product:

$$\langle f, g \rangle = \int_{\mathbb{R}} f(x) g(x) dx,$$

  and the corresponding norm is given by $\|f\| = \sqrt{\langle f, f \rangle} = \sqrt{\int_{\mathbb{R}} f(x)^2 dx}$.

- Like with vectors, we can consider forming more complex functions via additions of simpler ones. The following is an example of a multi-modal function constructed from three Gaussian-like bumps (by combining more and more bumps, we could clearly produce increasingly complicated functions):



- The property of orthonormality is also useful here: The functions $\{g_i\}$ are orthonormal if all $\langle g_i, g_j \rangle = 0$ and $\|g_i\| = 1$, and for any function in the span of the $\{g_i\}$ we can decompose

$$f(x) = \sum_i \langle f, g_i \rangle g_i(x).$$

  (Or if f is not in the span, then the right-hand side represents the *projection of f onto the span.*)

- At first one might think that the space spanned by sines and cosines is limited, but in fact these are rich enough to model *general piecewise continuous functions.* The relevant individual sines/cosines (or complex exponentials) will serve as a convenient orthonormal basis for such functions. (Unlike the vector case, the number of functions in the basis will be infinite, which is why we used the ambiguous notation $\sum_i$ above.)

We conclude this discussion by mentioning that allowing complex values will also be useful: For vectors $u, v \in \mathbb{C}^d$ the inner product is then $\sum_{i=1}^n u_i \overline{v}_i$ (with the bar meaning complex conjugation, $\overline{a + bi} = a - bi$), and the functions f, g the inner product is $\langle f, g \rangle = \int_{\mathbb{R}} f(x) \overline{g(x)} dx$, and the norm is $\|f\| = \sqrt{\int_{\mathbb{R}} |f(x)|^2 dx}$.

2 Fourier Series

Before proceeding, a quick note on terminology:

- The *period* T of a sin wave is the width of a full oscillation. The function $f(x) = \sin\left(\frac{2\pi x}{T}\right)$ has period T.
- The *(ordinary) frequency* ξ is the reciprocal of the period, $\xi = \frac{1}{T}$, e.g., giving $f(x) = \sin(2\pi\xi x)$.
- We will work more with the *angular frequency*, defined as $\omega = 2\pi\xi$, so that $f(x) = \sin(\omega x)$. Doing so has various pros and cons (e.g., removing the need to carry around 2π, but it does lead to having some "nuisance terms" later on).

In this section, we will consider functions produced by adding sines/cosines whose frequencies are *integer multiples* of some base frequency (e.g., $\sin(x)$, $\sin(2x)$, $\sin(3x)$, ...). This turns out to "only" be powerful enough to produce signals that have a regularly repeating pattern (periodicity), though that pattern itself can be highly complex. In Section 3 we will extend this idea to allow a continuum of frequencies (e.g., $\sin(\omega x)$ for arbitrary choices of ω), which will let us produce general functions that need not be periodic.

2.1 Using Trigonometric Functions (Sines and Cosines)

We first consider the space $\mathbb{S}$ of piecewise continuous functions $f : [-\pi, \pi] \to \mathbb{R}$. The specific interval $[-\pi, \pi]$ is considered for convenience, but this can be generalized to any finite interval $[a, b]$ We will also view these functions as being defined on the entire real line $\mathbb{R}$, where outside $[-\pi, \pi]$ the function is defined by having periodic behavior with period 2π. Since we are allowing complex values, we define the inner product for $\mathbb{S}$ as

$$\langle f, g\rangle = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)g(x)dx.$$

Notice the restriction to $[-\pi, \pi]$), and the division by π which is related to our use of angular frequency.

The Fourier Series is based on using the following basis for the space $\mathbb{S}$:

$$\left\{\frac{1}{\sqrt{2}}, \sin(x), \cos(x), \sin(2x), \cos(2x), ...\right\}. \tag{1}$$

We will omit proof that this is a basis, but make some comments on it in Appendix B. The orthonormality property can fairly easily be verified by direct integration.

Writing the basis elements in Eq. (1)as $\{e_n\}_{n=1}^{\infty}$, we can then decompose any function $f \in \mathbb{S}$ as:

$$f(x) = \sum_{n=1}^{\infty} \langle f, e_n\rangle e_n(x), \tag{2}$$

or substituting the functions explicitly:

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \big(a_n \cos(nx) + b_n \sin(nx)\big), \tag{3}$$

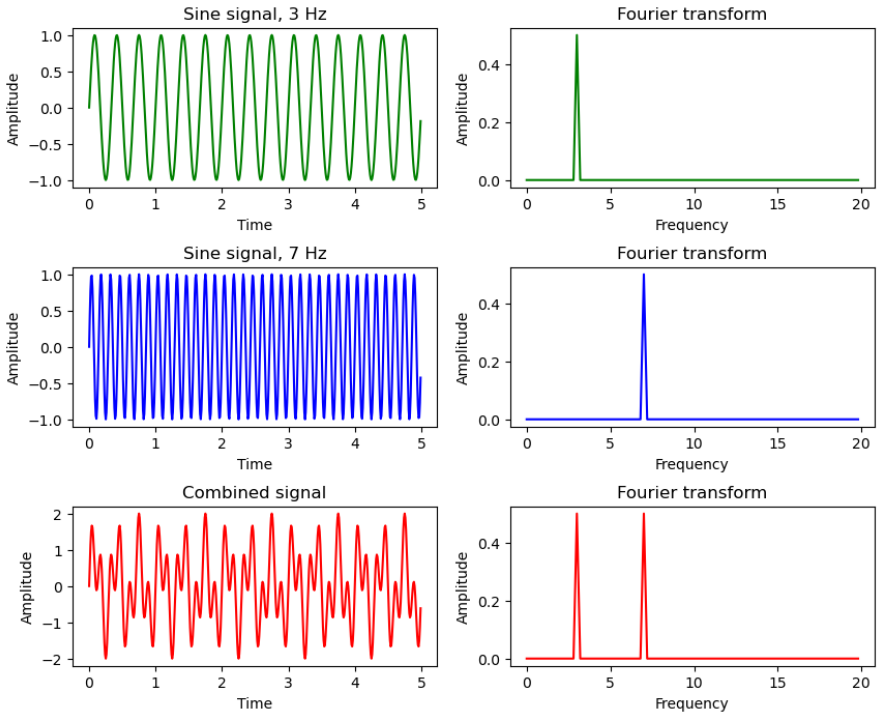


Figure 1: Illustration of two sine wave signals with different frequencies and their combined signal (first column), along with their Fourier transforms or decomposition into frequencies (second column).

where for $n = 1, 2, ...$

$$a_0 = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) dx$$
$$a_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \cos(nx) dx$$
$$b_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \sin(nx) dx.$$

Equation (3) is called the **Fourier series** expansion of f, and the terms a_n, b_n are called *Fourier coefficients*.

We can simplify this decomposition for certain classes of f. For example,

- For an even function f where $f(-x) = f(x)$, one can show that the b_n terms are all zero.
- Similarly, for an odd function f where $f(-x) = -f(x)$, one can show that the a_n terms are all zero.

We see that the Fourier decomposition can be viewed as a decomposition of a function f into a set of frequencies, mapped by the trigonometric basis functions. Some examples are shown in Fig. 1, where we see that signal comprising two sine signals with different frequencies can be readily decomposed into distinct frequencies. In general, functions that appear less 'smooth' with more fluctuations at smaller scales will tend to have larger high frequency coefficients (i.e., a_n and b_n for large n).

2.2 Using Complex Exponentials

The previous section had only considered real-valued periodic functions, and used sines/cosines in the choice of basis. Here we generalize to consider complex-valued functions, and adopt a basis based on complex exponentials, which perhaps surprisingly comes out to be *neater* and more compact to write. Since we now allow complex values, the preceding choice of inner product generalizes to

$$\langle f, g \rangle = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(x)\overline{g(x)}dx.$$

(Now we divide by 2π instead of π, which is related to complex numbers having 2 components.)

Similar to Eq. (1), we can introduce the following infinite set of functions:

$$\{1, e^{ix}, e^{-ix}, e^{i2x}, e^{-i2x}, ...\}. \tag{4}$$

We can again easily verify that these complex exponentials are orthogonal to one another by computing $\langle \psi_j, \psi_k \rangle = \int_{-\pi}^{\pi} e^{ijx} e^{-ikx} dx$ for each pair of functions (ψ_j, ψ_k) in the set. Similarly, we can verify that they have unit norm by writing $\|\psi\| = \sqrt{\langle \psi, \psi \rangle}$.

Then, the complex exponential Fourier series can be expressed as follows:

$$f(x) = \sum_{n=-\infty}^{\infty} c_n e^{inx}, \tag{5}$$

where for $n = 0, \pm 1, \pm 2, ...,$

$$c_n := \hat{f}_n = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(x) e^{-inx} dx. \tag{6}$$

Using Euler's formula $e^{i\theta} = \cos(\theta) + i\sin(\theta)$, we can relate the complex exponential Fourier series with that of the trigonometric Fourier series. Specifically, doing so gives the following:

- $c_n = \frac{a_n - ib_n}{2}$, $c_{-n} = \frac{a_n + ib_n}{2}$
- $a_n = c_n + c_{-n}$, $b_n = i(c_n - c_{-n})$
- Hence the symmetry properties: When f is an even function, we have $c_k = c_{-k}$ for $k = 1, 2, 3, ...,$ and similarly when f is an odd function, we have $c_k = -c_k$ for $k = 1, 2, 3...,$ zeroing out the odd sine and even cosine components accordingly.

2.3 Note on Finite-Term Approximation

In general, if we wanted to actually store the Fourier series of a function on a computer, we would have to truncate to a finite number of terms. Upon doing this, one might hope that a finite-term approximation of $f(x) = \sum_{n=-\infty}^{\infty} c_n e^{inx}$ is sufficient:

$$f(x) \overset{?}{\approx} f(x) = \sum_{n=-N}^{N} c_n e^{inx}$$

for suitably chosen N. It turns out that the approximation error is indeed very small for sufficiently "well-behaved" functions (e.g., ones with relatively smooth behavior and no abrupt changes).

On the other extreme, for discontinuous functions an important (and unfortunate) phenomenon known as the *Gibbs phenomenon* is observed, even when N is very large. The issue is that the basis functions

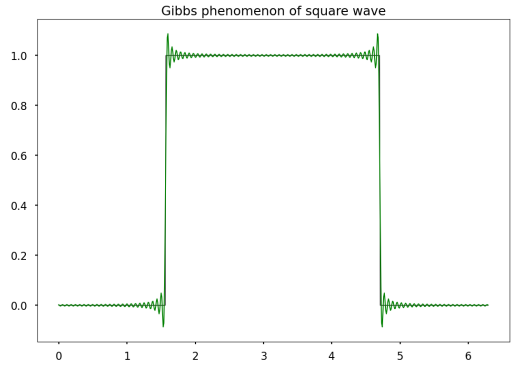


Figure 2: Illustration of Gibbs phenomenon for a square wave. Notice the "ringing" pattern near the discontinuities.

(sines and cosines) are all continuous, so any finite combination of them will also be continuous. As a result, when we truncate the infinite series, we get a "ringing" pattern causing large approximation errors near the discontinuity; see Figure 2 for an example using the square wave. By comparison, further away from the discontinuities, the approximation error is very small.

2.4 Parseval's Theorem and Energy Interpretation

The following result is known as the *Parseval's theorem* (or sometimes *Rayleigh's identity*):

$$\|f\|^2 = \sum_{n=-\infty}^{\infty} |\hat{f}_n|^2 = \sum_{n=-\infty}^{\infty} |\left\langle f, e^{inx} \right\rangle|^2 \tag{7}$$

This roughly states that *the energy (i.e., sum or integral of the squared values) of a signal is the same as the energy of its Fourier transform.*

To show this property, letting e_n denote the n-th Fourier basis function, we have

$$\begin{aligned}
\|f\|^2 &= \langle f, f \rangle \qquad \text{(definition of norm)} \\
&= \left\langle \sum_{n=-\infty}^{\infty} \langle f, e_n \rangle e_n, \sum_{m=-\infty}^{\infty} \langle f, e_m \rangle e_m \right\rangle \quad \text{(Fourier expansion of } f\text{)} \\
&= \sum_{n,m=\infty}^{\infty} \langle f, e_n \rangle \overline{\langle f, e_m \rangle} \langle e_n, e_m \rangle \quad \text{(linearity of inner product)} \\
&= \sum_{n=\infty}^{\infty} \langle f, e_n \rangle \overline{\langle f, e_n \rangle} \quad \text{(orthonormality of basis)} \\
&= \sum_{n=-\infty}^{\infty} |\left\langle f, e^{inx} \right\rangle|^2. \quad \text{(since } a\overline{a} = |a|^2\text{)}
\end{aligned} \tag{8}$$

Note also that in view of this property, we can interpret $|\left\langle f, e^{inx} \right\rangle|^2$ as the amount of energy associated with the n^{th} frequency.

3 Fourier Transform

Thus far we have only looked at periodic functions (or alternatively, functions restricted to a finite interval, $[-\pi, \pi]$). In this section, we move to general functions from $\mathbb{R} \to \mathbb{C}$, using the periodic case as a building block. Specifically, we consider a general function $f : \mathbb{R} \to \mathbb{C}$ having the following two properties: (i) it is piecewise continuous on every finite interval, and (ii) it is absolutely integrable, i.e., $\int_{-\infty}^{\infty} |f(x)|dx < \infty$. (In fact, we will not always require (ii) to hold, e.g., we will still talk about the Fourier transform of a sinusoid.)

The idea will be to apply the Fourier series to the function restricted to a finite range $[-L, L]$ (for some $L > 0$), and then consider what happens when $L \to \infty$. We start with the following observations:

- For any $L > 0$, by generalizing the case that $L = \pi$ (handled above), we can derive the Fourier series $f(x) = \sum_{n=-\infty}^{\infty} c_{n,L} e^{i\omega_n x}$, where $\omega_n = n\pi/L$ and $c_{n,L} = \frac{1}{2L} \int_{-L}^{L} f(x) e^{-i\omega_n x} dx$.
- Observe that each frequency is a multiple of π/L. We thus define $\Delta_{\omega,L} = \pi/L$, which gives $\omega_n = n\Delta_{\omega,L}$, and the Fourier coefficient can be expressed as

$$c_{n,L} = \Delta_{\omega,L} \cdot \frac{1}{2\pi} \cdot \underbrace{\int_{-L}^{L} f(x) e^{-i\omega_n x} dx}_{\text{Denoted by } \hat{f}_L(\omega_n)}. \tag{9}$$

Now consider the limiting case where $L \to \infty$. Then, the integration over $[-L, L]$ approaches integration over the whole real line,[1] and we also have $\Delta_{\omega,L} \to 0$. This means that the summation defining $f(x)$ approaches an integral:

$$f(x) = \lim_{L\to\infty} \sum_{n=-\infty}^{\infty} c_{n,L} e^{i\omega_n x} \tag{10}$$

$$= \lim_{L\to\infty} \frac{1}{2\pi} \sum_{n=-\infty}^{\infty} \hat{f}_L(\omega_n) e^{i\omega_n x} \Delta_{\omega,L} \tag{11}$$

$$= \frac{1}{2\pi} \int_{-\infty}^{\infty} \hat{f}(\omega) e^{i\omega x} d\omega, \tag{12}$$

where $\hat{f}(\omega) = \int_{-\infty}^{\infty} f(x) e^{-i\omega x} dx$.

The above analysis leads us to define to **Fourier Transform** and **Inverse Fourier Transform** as follows:

$$\hat{f}(\omega) = \mathcal{F}(f(x)) = \int_{-\infty}^{\infty} f(x) e^{-i\omega x} dx \tag{13}$$

$$f(x) = \mathcal{F}^{-1}(\hat{f}(\omega)) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \hat{f}(\omega) e^{i\omega x} d\omega. \tag{14}$$

This pair of operators are the counterpart to the Fourier series for non-periodic functions. The transform $\mathcal{F}(\cdot)$ can still be viewed as a change of basis from a given "natural basis" to the "frequency basis". Notice that now *all* frequency values $\omega \in \mathbb{R}$ play a role, whereas for the Fourier series it was only integer multiples of a base frequency. Accordingly, we now get $f(x)$ by integrating over all ω, rather than summing over integers n.

As we will see soon, there are several properties in the frequency domain that can be exploited to be able to solve problems that would be challenging in the original domain.

[1] This is formally justified by the condition $\int_{-\infty}^{\infty} |f(x)|dx < \infty$, which we mentioned above.

Note on ordinary frequency vs. angular frequency. The quantity ω above is the *angular frequency*, and is related to the *ordinary frequency* ξ via $\omega = 2\pi\xi$. The distinction is just a minor matter of normalization, but it's useful to be aware of the three main variations of the Fourier transform:

- The one above is called non-unitary with angular frequency.
- The unitary version with angular frequency puts a factor $\frac{1}{\sqrt{2\pi}}$ in the expressions for $\hat{f}$ and f, rather than putting the entire 2π in the expression for f. This makes the equations "more symmetric" but can lead to a lot of "niusance" $\frac{1}{\sqrt{2\pi}}$ factors.
- The version with ordinary frequency puts 2π in the exponent instead, which turns out the remove the need for any pre-factor: $\hat{f}(\xi) = \int_{-\infty}^{\infty} f(x)e^{-i2\pi\xi x}dx$ and $f(x) = \int_{-\infty}^{\infty} \hat{f}(\xi)e^{i2\pi\xi x}d\xi$.

Very brief note on multi-input functions: The Fourier transform can also be defined for functions with inputs in $\mathbb{R}^d$, with $\mathbb{R}^2$ being particularly important in image processing. We will stick to single-input functions in this lecture, but the main properties generally carry over naturally.

4 Examples and Properties

Here we overview some examples and main properties of the Fourier transform – see the tables are `https://en.wikipedia.org/wiki/Fourier_transform#Tables_of_important_Fourier_transforms` for a useful summary and a more detailed list.

Before proceeding, we need to introduce a "strange but useful" notion called the *Dirac delta function* $\delta(x)$, which satisfies the following equality:

$$\delta(x) = \begin{cases} \infty & x = 0 \\ 0 & x \neq 0. \end{cases}$$

To make this more precise, we impose the following "definition" or "convention":

$$\int_{-\infty}^{\infty} \delta(x)\, dx = 1. \tag{15}$$

We can think of it in the following way: Imagine a rectangle-shaped function whose value is $\frac{1}{w}$ in the interval $[-w/2, w/2]$ for some $w > 0$, and 0 elsewhere. This is a perfectly standard function that we could work with in the usual way. The Dirac delta function captures how such a function behaves *in the limit as* $w \to 0$ – thus, it is an infinitely narrow and infinitely high "spike" but it still integrates to one.

For any function $f(x)$, it follows from (15) that

$$\int_{-\infty}^{\infty} \delta(x-a)f(x)\, dx = f(a). \tag{16}$$

The Dirac delta function is useful when studying Fourier transforms (and also for Linear Time Invariant systems, which we won't cover in detail), as two of the examples below demonstrate.

4.1 Examples

Some sketches of common functions and their Fourier transform pairs are given in Fig. 3. We give the details of one example (the rectangular function); the interested student may want to try some of the others (using Eq. (13) and Eq. (14)) and/or consult the Wikipedia page for a longer list of examples.

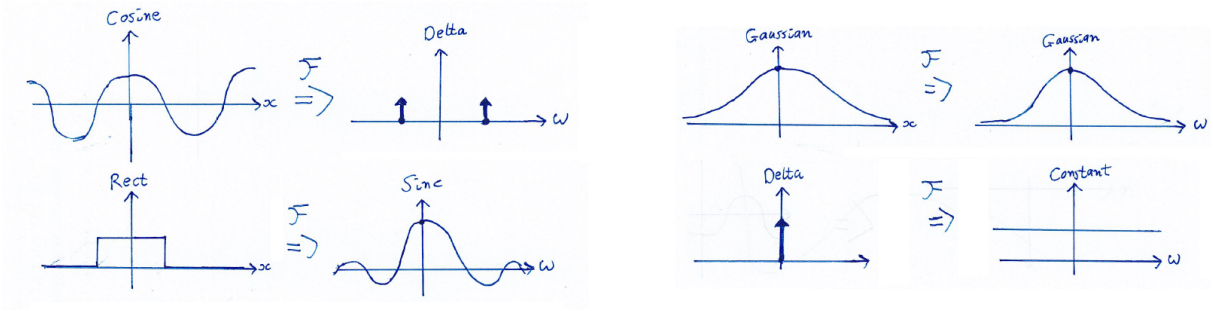


Figure 3: Examples of common functions and their Fourier transform pairs.

Consider the rectangular function

$$f(x) = \begin{cases} 1 & |x| \leq \frac{1}{2} \\ 0 & \text{otherwise.} \end{cases}$$

The Fourier transform is given by

$$\begin{aligned} \hat{f}(\omega) &= \int_{-1/2}^{1/2} e^{-i\omega x} dx \\ &= \Big[\frac{1}{-i\omega} e^{-i\omega x}\Big]_{-1/2}^{1/2} \\ &= \frac{e^{-i\omega/2} - e^{i\omega x/2}}{-i\omega} \\ &= \frac{\sin(\omega/2)}{\omega/2}, \end{aligned}$$

where the last line uses $\sin x = \frac{e^x - e^{-x}}{2i}$ (from Euler's identity). Then, introducing the notation $\text{sinc}(x) = \frac{\sin(\pi x)}{\pi x}$ (the *sinc function*), we get

$$\hat{f}(\omega) = \text{sinc}\Big(\frac{\omega}{2\pi}\Big).$$

4.2 Linearity

It is straightforward to see that the Fourier transform is a linear operator, i.e., $\mathcal{F}[af(x) + bg(x)] = a\mathcal{F}[f(x)] + b\mathcal{F}[g(x)]$, given that the integral operators are linear. This applies similarly for the inverse Fourier transform.

4.3 Shifting Properties

The following properties are fairly straightforward to prove using the definition of the (inverse) Fourier transform and standard integration techniques like change of variable and/or $e^{a+b} = e^a e^b$: If $f(x)$ has Fourier transform $\hat{f}(\omega)$, then

- For any constant a, the function $f(ax)$ has Fourier transform $\frac{1}{|a|}\hat{f}(\omega/a)$. This means that when a function gets narrower in the original domain (i.e., $a > 1$) it gets broader in the frequency domain, and vice versa. This is intuitive, because making a function narrower (while otherwise maintaining the same shape) means that any function increases/decreases get sharper/steeper, and faster changes correspond to higher frequencies.

- For any constant c, the function $f(x-c)$ has Fourier transform $\hat{f}(x)e^{-ic\omega}$. Similarly, the function $f(x)e^{icx}$ has Fourier transform $\hat{f}(\omega-c)$. That is, a shift in one domain translates to multiplication by a complex exponential in the other.

4.4 Convolution Properties

We can also consider the convolution operation between two functions f and g :

$$(f*g)(x) = \int_{-\infty}^{\infty} f(x-y)g(y)dy. \tag{17}$$

Intuitively, we can view convolution as a kind of smoothing or averaging operation. It is a very important concept in both theory and practice; for example:

- We will briefly discuss how it is used in the context of understanding linear time-invariant (LTI) systems in Section 6.2.
- The Fourier transform applied to a probability density function is known as the *characteristic function*. In this context, the relevance of convolution is that if X, Y are independent with density functions f_X, f_Y, then the random variable $Z = X + Y$ has density function $f_Z = f_X * f_Y$.

A very useful property of the Fourier transform is that *convolution in the original domain corresponds to multiplication in the Fourier domain*:

$$\mathcal{F}(f*g) = \mathcal{F}(f)\mathcal{F}(g). \tag{18}$$

That is, the Fourier transform of $(f*g)(x)$ is $\hat{f}(\omega)\hat{g}(\omega)$.

We can see this by considering the inverse Fourier transform of $\hat{f}\hat{g}$:

$$\begin{aligned}
\mathcal{F}^{-1}[\hat{f}\hat{g}](x) &= \frac{1}{2\pi}\int_{-\infty}^{\infty} \hat{f}(\omega)\hat{g}(\omega)e^{i\omega x}d\omega \qquad \text{(by definition of } \mathcal{F}^{-1}) \\
&= \frac{1}{2\pi}\int_{-\infty}^{\infty} \hat{f}(\omega)\int_{-\infty}^{\infty} g(y)e^{-i\omega y}dy\; e^{i\omega x}d\omega \quad \text{(by definition of } \hat{g}) \\
&= \int_{-\infty}^{\infty} g(y)\underbrace{\frac{1}{2\pi}\int_{-\infty}^{\infty} \hat{f}(\omega)e^{i\omega(x-y)}d\omega}\, dy \quad \text{(by swapping order of integrals)} \qquad (19) \\
&= \int_{-\infty}^{\infty} g(y)f(x-y)dy \quad \text{(by formula for inverse FT)} \\
&= f*g. \qquad (20)
\end{aligned}$$

4.5 Fourier Transform vs. its Inverse

Comparing the formula for the Fourier transform and its inverse in (13) and (14), we see that the two are nearly identical – the only differences are (i) the $\frac{1}{2\pi}$ factor (this difference even disappears in the other version of the Fourier transform discussed after (14)), and (ii) the use of $-i$ vs. i in the exponent. In this sense, the Fourier transform and inverse Fourier transform are "essentially" doing the same thing.

For this reason, when we see a property in the original domain implying some other property in the frequency domain, the role of the domains can easily be reversed. For example:

- Convolution in the original domain translates to multiplication in frequency domain;

- Conversely, multiplication in the original domain translates to convolution in the frequency domain.

(Care may just be needed in getting certain signs and scaling factors correct.)

4.6 Derivative Properties

How we consider how the Fourier transform of a differentiable function $f(x)$ relates to the Fourier transform of its derivative $f'(x)$. This turns out to be very useful for solving differential equations; see Section 6.3 for an example.

Technically we need certain smoothness/continuity and absolutely integrable conditions, but we will not go into the details of those, and instead simply state that we can get the following using integration by parts:

$$\begin{aligned}
\mathcal{F}(f'(x)) &= \int_{-\infty}^{\infty} \underbrace{f'(x)}_{dv} \underbrace{e^{-i\omega x}}_{u} dx \\
&= \underbrace{\left[f(x)e^{-i\omega x}\right]_{-\infty}^{\infty}}_{0} - \int_{-\infty}^{\infty} f(x)(-i\omega e^{-i\omega x})dx \\
&= i\omega \underbrace{\int_{-\infty}^{\infty} f(x)(e^{-i\omega x})dx}_{\mathcal{F}(f(x)} \\
&= i\omega \mathcal{F}(f(x)). \qquad (21)
\end{aligned}$$

We can similarly take the n-th order derivative $f^{(n)}$, and obtain $\mathcal{F}(f^{(n)}) = (i\omega)^n \mathcal{F}(f(x))$.

Hence, the derivative of a function, when considered after a Fourier transform to the frequency domain, will just become multiplication or scaling by $i\omega$ in the frequency domain; in particular, the magnitude at frequency ω will get scaled by ω (analogous to how $\frac{d}{dx}\sin(\omega x) = \omega\cos(\omega x)$).

4.7 Parseval's Theorem

Parseval's theorem applies to the Fourier transform as well, and is stated as follows:

$$\int_{-\infty}^{\infty} |f(x)|^2 dx = \frac{1}{2\pi}\int_{-\infty}^{\infty} |\hat{f}(\omega)|^2 d\omega.$$

In other words, the Fourier transform preserves the (squared) L2-norm, up to a constant related to the normalization of the basis (in this case $\frac{1}{2\pi}$). Again, the intuition is that the total energy of the time signal equals the total energy of the frequency signal.

5 Discrete Time

Thus far, we have considered functions $f(x)$ with a real-valued input $x \in \mathbb{R}$. Naturally, when we store a signal on a computer, we need to use *finitely many bits*, and a natural approach is to only consider a finite number of x values. (Each $f(x)$ will also need to be "quantized" to finitely many bits, but usually that's less of an issue because 64-bit numbers tend to be "accurate enough".) In addition, some signals/functions are simply discrete in nature to begin with, and we would still like to have a notion of Fourier transform for them.

For concreteness, we will think of the function input as representing "time", even though in applications it could represent other dimensions such as spatial. In addition, we will switch notation (with apologies if it is confusing) to choices that are more common when working in discrete time:

- The input x will be renamed to t, representing "time";
- Instead of a function $f(\cdot)$ with Fourier transform $\hat{f}(\cdot)$, we will consider a function $x(\cdot)$ with Fourier transform $X(\cdot)$.
- We will shortly switch from continuous time ($t \in \mathbb{R}$) to discrete time ($n \in \mathbb{Z}$), and in discrete time we will use square brackets to highlight this distinction, e.g., $x[n]$.
- Instead of the angular frequency ω, we will consider the ordinary frequency ξ (they are related by $\omega = 2\pi\xi$). As we mentioned previously, using ordinary frequency, the continuous-time Fourier transform of a signal $x(t)$ is

$$X(\xi) = \int_{-\infty}^{\infty} x(t) e^{-i2\pi\xi t}\, dt. \tag{22}$$

5.1 Sampling of Continuous Signals

It is useful to first think about the following: If we can measure some real-valued signal $x(t)$ by querying various t values, how should we do this in a manner that lets us store it on a computer but still retain the relevant information? By either stopping the measurements at some point or working in blocks of a given length, we can assume that t is bounded, say $t \in [0, T]$.

The natural strategy is to query $x(\cdot)$ at regular intervals, say $\{0, T_s, 2T_s, 3T_s, \ldots, T\}$ for some *sampling interval* T_s (which we'll assume to be such that $\frac{T}{T_s}$ is an integer):

$$x[n] = x(t)\big|_{t=nT_s}. \tag{23}$$

This gives us a discrete-time signal of length $\frac{T}{T_s} + 1$. A key question is then what choice of T_s to use. At first glance, it seems that any $T_s > 0$ leads to *some* loss of information, with the loss vanishing only in the limit as $T_s \to 0$. Remarkably, as long as $x(t)$ does not contain arbitrarily high frequencies, we can in fact have *zero loss* for a carefully chosen T_s. It is more natural to state the result in terms of $\xi_s = \frac{1}{T_s}$, which we call the *sampling frequency*.

Shannon-Nyquist Sampling Theorem: *Let $x(t)$ be a continuous-time signal, let $X(\xi)$ be its Fourier transform, and let B be the highest frequency for which $X(\xi) \neq 0$ (i.e., $X(\xi) = 0$ for all $\xi > B$. The letter B stands for "bandwidth"). Let $x[n]$ be the discrete-time signal formed via (23). Then, $x(t)$ can be* perfectly reconstructed *from $x[n]$ provided that the sampling frequency ξ_s satisfies $\xi_s > 2B$.*

This is a very famous result in signal processing, but we will skip its proof. In short, moving to discrete time loses nothing (at least mathematically) if we sample above twice the highest frequency of the signal.

(Optional**) Aliasing.** It is also useful to understand what happens if $\xi_s < 2B$ (which is unavoidable if $B = \infty$; not all signals are bandlimited). To answer this, it is useful to think of all $t \in \mathbb{R}$ rather than only $t \in [0, T]$ (e.g., by taking $x(t) = 0$ outside $[0, T]$). Then $x[n]$ from (23) is defined for all integers. If we use the formula for the Fourier transform on $x[n]$, but replace the integral by a sum since $x[n]$ is discrete-time, we get

$$X_s(\xi) = \sum_{n=-\infty}^{\infty} x[n] e^{-i2\pi\xi n T_s}. \tag{24}$$

Alternatively, this can be interpreted as the Fourier transform of $x_s(t)$, defined to have a delta Dirac function of height $x[n]$ whenever $t = nT_s$ for some integer n. Using the linearity of the Fourier transform and the fact that the Dirac function's Fourier transform is a complex exponential, this interpretation can be shown to lead to the following (details omitted):

$$X_s(\xi) = \sum_{n=-\infty}^{\infty} X(\xi - n/T_s) = \sum_{n=-\infty}^{\infty} X(\xi - n\xi_s),$$

meaning that the spectrum of $X_s(\cdot)$ is a sum of *all shifted copies* of $X(\cdot)$, where the shifts are by $\pm\xi_s, \pm 2\xi_s$, and so on. If $\xi_s < 2B$, these shifted copies start to overlap, a phenomenon known as *aliasing*:

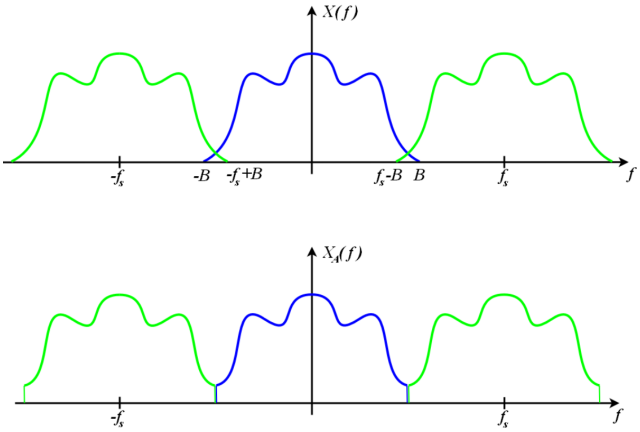


Figure 4: Illustration of aliasing in the frequency domain. (Frequencies here are denoted by f instead of ξ.)

The issue is that high frequency signals look the same as low-frequency ones when we sample too slowly, an example is shown as follows (this image and the one above are from Wikipedia):

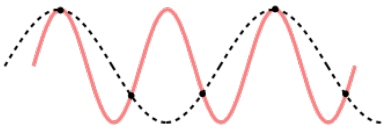


Figure 5: Illustration of aliasing in the time domain.

The effect of this is that any frequency $\xi \geq \frac{\xi_s}{2}$ gets "folded down" and looks the same as a lower frequency (e.g., a signal of frequency exactly ξ_s is indistinguishable from a signal of frequency 0, i.e., a constant signal). Having said this, if the frequency content of $X(\xi)$ is small enough for $\xi \geq \frac{\xi_s}{2}$, the impact of aliasing may be negligible.

5.2 Discrete Fourier Transform (DFT)

We are now ready to define the Discrete Fourier Transform (DFT). One way to view this is to start with $X_s(\xi)$ from (24) and simply evaluate it at a finite number (denoted by N) of frequencies:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-i\frac{2\pi}{N}kn}, \quad k = 0, 1, 2, ..., N-1. \tag{25}$$

This is another kind of sampling, but now it is *sampling in frequency domain* (with intervals of length $\frac{\xi_s}{N}$) rather than in time domain. With this definition, it can be shown that the following *inverse DFT* recovers the discrete-time signal:

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] e^{i \frac{2\pi kn}{N}}, \quad n = 0, \ldots, N-1. \tag{26}$$

Observe that both $x[n]$ and $X[k]$ are now described by just N numbers, making them amenable to storage and processing on a computer. The following should be noted:

- Any information about $x(t)$ outside $t \in [0, (N-1)T_s]$ is lost (though we could of course form a separate DFT corresponding to the later segments of the signal). Choosing N to make this consistent with $t \in [0, T]$, we get $N = \frac{T}{T_s} + 1$. (The $+1$ term may be avoided by instead sampling at the *midpoints* of length-T_s regions, but we won't worry about this.)
- Aliasing may also incur a loss of information, as discussed above.

As we hinted earlier, the DFT is also useful for signals that are simply discrete-valued to begin with (rather than sampled version of continuous ones). In such scenarios, the length N is simply the length of the signal we are given (or perhaps the length of sub-blocks of the signal that we opt to work with).

The DFT has analogs of most of the main properties of the continuous-time Fourier transform (e.g., convolution property, Parseval's theorem, etc.). See `https://en.wikipedia.org/wiki/Discrete_Fourier_transform#Properties` for a summary.

5.3 Fast Fourier Transform (FFT)

The DFT consists of computing N coefficients, each defined as a sum over N terms, so naively the computation time is $O(N^2)$. (Or treating $x[n]$ as a vector, we can view the DFT as multiplication by an $N \times N$ "DFT matrix".) It turns out that the naive approach is actually doing a lot of repeated/redundant computation. The *Fast Fourier Transform* (FFT) is a (very) famous algorithm that avoids this, and brings the computation time down to $O(N \log N)$. There are several versions of the FFT, and the one that we present here is due to Cooley and Tukey.

The general procedure of Cooley and Tukey's algorithm is to re-express the discrete Fourier transform (DFT) recursively (divide-and-conquer), to reduce the computation time. We first rewrite the DFT to be

$$X[k] = \sum_{k=0}^{N-1} x[n] W_N^{kn}, \tag{27}$$

where

$$W_N^{kn} = e^{-i \frac{2\pi kn}{N}}. \tag{28}$$

It is useful to note that W_N^{kn} has a periodicity property: $W_N^{(k+N)n} = W_N^{kn}$.

To develop the FFT, we first split the single summation over N samples into 2 summations, each with $\frac{N}{2}$ samples, one for n even and the other for n odd. Substituting $m = \frac{n}{2}$ for n even and $m = \frac{n-1}{2}$ for n odd, we have

$$X[k] = \sum_{m=0}^{\frac{N}{2}-1} x[2m] W_N^{2mk} + \sum_{m=0}^{\frac{N}{2}-1} x[2m+1] W_N^{(2m+1)k}. \tag{29}$$

To simplify this, observe that

$$W_N^{2mk} = e^{-i\frac{2\pi(2mk)}{N}} = e^{-i\frac{2\pi mk}{\frac{N}{2}}} = W_{\frac{N}{2}}^{mk}. \tag{30}$$

Therefore,

$$X[k] = \sum_{m=0}^{\frac{N}{2}-1} x[2m] W_{\frac{N}{2}}^{mk} + W_N^k \sum_{m=0}^{\frac{N}{2}-1} x[2m+1] W_{\frac{N}{2}}^{mk}. \tag{31}$$

Defining $G[k]$ to be the length-$N/2$ DFT on even indices, and $H[K]$ the one on odd indices, it follows that

$$X[k] = G[k] + W_N^k H[k]. \tag{32}$$

Thus the N-point DFT $X[k]$ can be obtained from two $\frac{N}{2}$-point transforms! Although the frequency index k ranges over N values, only $\frac{N}{2}$ values of $G[k]$ and $H[k]$ actually need to be computed, since $G[k]$ and $H[k]$ are periodic with period $\frac{N}{2}$. An illustration is given in Figure 6.

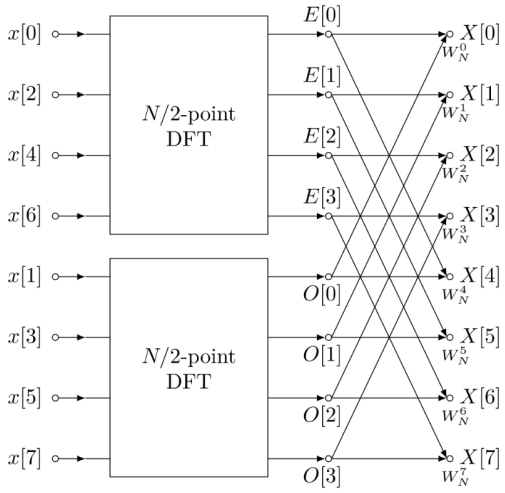


Figure 6: Calculating the N-point DFT from the $N/2$-point DFT. (Image from `https://lutpub.lut.fi/handle/10024/164024`)

Supposing that N is a power of 2, we can repeat the above process on the two $\frac{N}{2}$-point transforms, breaking them down to $\frac{N}{4}$-point transforms, and so on, until we come down to 2-point transform. At a recursion depth of $\log_2 N$ we are down to a length of 1, which is trivial. The $O(N \log N)$ scaling then simply comes from performing $O(N)$ computation at each depth.

6 (**Optional**) Applications

The Fourier transform is extensively used throughout signal processing, communications, machine learning, theoretical computer science, statistics, and more. We give just a few examples of applications here, and we don't go into much detail

6.1 Signal Denoising

A fundamental task in signal processing is *denoising* – removing as much noise as possible while keeping as much of the underlying signal as possible. The Fourier transform provides a very natural view for this task –

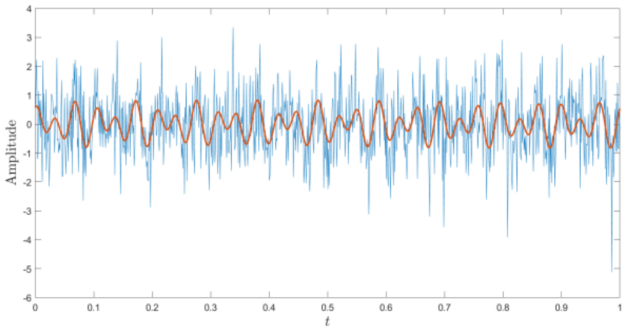


Figure 7: Signal (red) contaminated with Gaussian white noise (blue shows the noisy signal).

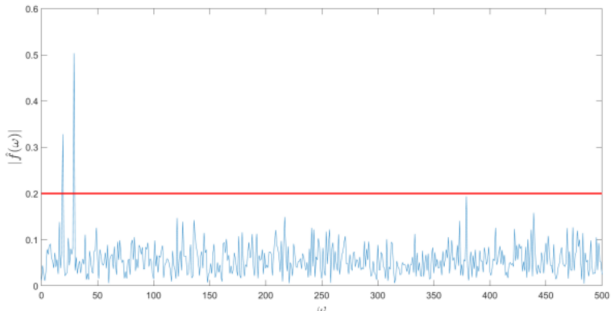


Figure 8: Frequency spectrum of the noisy signal after Fourier transform.

if we know that the signal lies (mostly) in a certain frequency band, then we can filter out other frequencies and thus attenuate or remove any noise that occurs there. (Of course, any noise at the same frequencies as the signal will remain, but that tends to be a relatively small amount.)

Figure 7 gives an example of a signal corrupted by noise. The noise makes the signal highly corrupted, with very erratic oscillations. But since the noise fluctuates so much but the signal only exhibits mild oscillations, this suggests that the *signal lies at low frequencies* whereas the *noise contains high frequencies* that we should try to remove. The Fourier transform lets us see this much more precisely – see Figure 8. With this picture in mind, a natural denoising approach is to only keep frequencies whose value exceeds some thresold (e.g., the red line in the figure). Other methods are also possible, like keeping *all* low frequencies. There are still various design issues like what threshold or frequency range to use, or what prior knowledge (if any) we have about the signal, etc., but overall this is a very natural and effective approach to denoising.

6.2 Linear Time Invariant Systems

In signal processing systems, we are interested in operations that input one signal $x(t)$ (or $x[n]$ in discrete time) and output another signal $y(t)$. (In fact, denoising as described above would be an example this!)

An important class of these is *linear time-invariant* (LTI) systems, which satisfy the linearity property (an input of $ax_1(t) + bx_2(t)$ gives output $ay_1(t) + by_2(t)$) and the time invariance property (an input of $x(t-\tau)$ gives output $y(t-\tau)$).

LTI systems are fully described by their *impulse response*, which is the output when $x(t) = \delta(t)$ (the Dirac delta function) – essentially, giving the system a "flick" and seeing how it responds. Once the impulse response is known, the output associated with an arbitrary input $x(t)$ is the convolution of x and the impulse response. (You may wish to try to prove this using the LTI properties and (15).)

By moving to the Fourier view, the convolution gets replaced by *multiplication*, which can be much easier to understand, interpret, and design based on.

6.3 Solving Differential Equations

The Fourier transform is a useful tool for solving differential equations efficiently, e.g., in the study of dynamical systems and control systems. We demonstrate one example below.

In this example, we consider a damped harmonic oscillator shown in Fig 9, which is a particle of mass m subject to a spring force and a damping force.

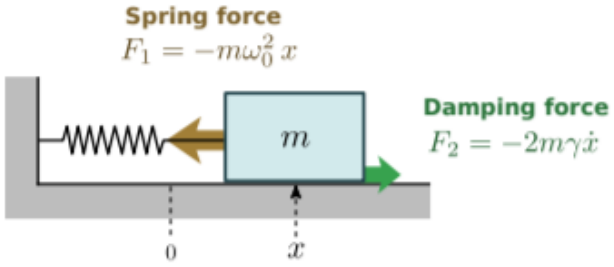


Figure 9: Illustration of a Damped Harmonic Oscillator

The motion of the particle can be derived using Newton's second law to obtain the ordinary differential equation (ODE):

$$\frac{d^2x}{dt^2} + 2\gamma y\frac{dx}{dt} + \omega_0^2 x(t) = 0. \tag{33}$$

Moreover, if an additional *driving force* $f(t)$ is introduced that operates on the particle, we can further generalize this to

$$\frac{d^2x}{dt^2} + 2\gamma\frac{dx}{dt} + \omega_0^2 x(t) = \frac{f(t)}{m}. \tag{34}$$

To solve for $x(t)$, we first take the Fourier transform of both sides of the above equation. The result is

$$-\omega^2\hat{x}(\omega) - 2i\gamma\omega\hat{x}(\omega) + \omega_0^2\hat{x}(\omega) = \frac{\hat{f}(\omega)}{m}, \tag{35}$$

where $\hat{x}(\omega)$ and $\hat{f}(\omega)$ are the Fourier transforms of $x(t)$ and $f(t)$ respectively.

Thanks to moving to the Fourier domain, the inconvenient derivative operations are replaced by simpler and easier-to-analze operations. Specifically, we have obtained an algebraic equation that can easily be solved:

$$X(\omega) = \frac{F(\omega)/m}{-\omega^2 - 2i\gamma\omega + \omega_0^2}. \tag{36}$$

Knowing $X(\omega)$, we can use the inverse Fourier transform to obtain $x(t)$ (which is not an especially "simple" or "nice" expression, but it does give the solution we are after, and it can readily be computed numerically):

$$x(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty}\frac{F(\omega)/m}{-\omega^2 - 2i\gamma\omega + \omega_0^2}e^{-i\omega t}d\omega, \quad \text{where } F(\omega) = \int_{-\infty}^{\infty} e^{i\omega t}f(t)dt. \tag{37}$$

This gives the system behavior of the particle under the effect of the external force. In control system design, we can further design the $f(t)$ at ease to realize the desired system behavior.

To summarize, the solution procedure for the driven harmonic oscillator equation consists of (i) using the Fourier transform on $f(t)$ to obtain $F(\omega)$, (ii) using the above equation to find $X(\omega)$ algebraically, and (iii) performing an inverse Fourier transform to obtain $x(t)$. We mainly leverage the derivative property of the Fourier transform to solve ODEs in real applications.

6.4 Fast Matrix Multiplication

Many problems (including some of those above) can be understood via the following picture:

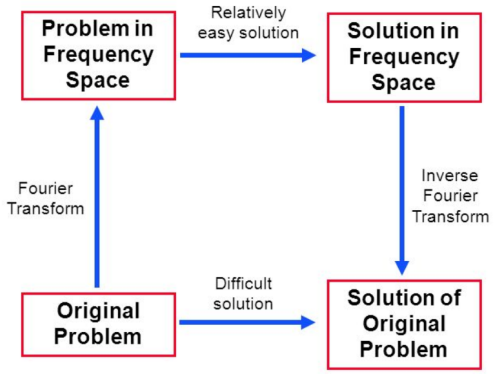


Figure 10: Problem-solving using Fourier transform

Notably, taking the 3-step approach (to Fourier, solve, and then inverser Fourier) can be much easier than the 1-step approach. Here we give an example in the context of matrix multiplication, in which case it's the *discrete Fourier transform* that comes into play.

For an $n \times n$ matrix A and a $n \times 1$ vector x, the computation of Ax takes $O(n^2)$ time, and in general this can't be improved (since just reading A takes $O(n^2)$ time). But it can be improved for certain classes of matrices. One such class is *circulant matrices*, which take the form

$$\begin{bmatrix} c_0 & c_1 & \dots & c_{n-2} & c_{n-1} \\ c_{n-1} & c_0 & c_1 & \dots & c_{n-2} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ c_1 & c_2 & \dots & c_{n-1} & c_0 \end{bmatrix}$$

That is, the rows are cyclic shifts of each other, and we always have 'diagonals' taking a common value. If we let F be the $n \times n$ matrix that performs the DFT operation, and collect then c_i values into a vector c, then it can be shown that $A = F^{-1}\text{diag}(Fc)F$.[2] Thus, we can compute Ax in $O(n \log n)$ time by 3 invocations of the FFT algorithm (two for F and one for F^{-1}), and using the fact that multiplication by a diagonal matrix only takes $O(n)$ time.

A circulant matrix is in fact performing a kind of convolution operation (with "wrap around" from index $n+1$ back to 1), so this example is essentially a different way of stating that convolution becomes multiplication after applying the (discrete) Fourier transform.

[2] In more technical terms, this means that *the DFT matrix diagonalizes A*.

6.5 Other

Other applications of the Fourier transform include (but are not limited to) the following:

- In the field of communication systems, FT facilitates the modulation and demodulation of signals, thereby playing a crucial role in signal transmission and reception.
- It also finds significant usage in image and audio processing, aiding in the analysis, manipulation, and compression of image and sound signals.
- In machine learning, FT is used for various tasks such as feature encoding, fast calculation, modeling smoothness assumptions in optimization, and so on.
- As hinted at the start of the lecture, a more abstract use is the analsyis of Boolean functions (e.g., see `https://www.youtube.com/watch?v=lXJP-UkTl-4`), and there are many others.

Appendix

A (**Optional**) More Detailed Background on Bases

Here we review in more detail some mathematical foundations and core concepts associated with vector spaces and the notion of basis. We assume that the reader is familiar with some linear algebra and concepts such as vector spaces, linear independence, and span.

- **Basis.** A basis is a set of vectors $\{v_i\}$ (for a vector space $\mathbb{V}$) if $\{v_i\}$ are linearly independent and $\mathbb{V} = \text{span}\{v_i\}$. For continuous vector spaces (e.g., $\mathbb{R}^d$), there are infinitely many possible bases.
- **Inner product.** The inner product can be thought of as a generalization of the dot product of Euclidean space. Given $\mathbb{V}$ that is a complex linear space and vectors $u, v \in \mathbb{V}$, the inner product $\langle u, v\rangle \in \mathbb{C}$ satisfies the following properties:
  - $\langle v, v\rangle \geq 0$
  - $\langle v, v\rangle = 0$ iff $v = 0$
  - $\forall u, v, w \in \mathbb{V}$ and $a, b \in \mathbb{C}, \langle au + bv, w\rangle = a\langle u, w\rangle + b\langle v, w\rangle$
  - $\langle u, v\rangle = \overline{\langle v, u\rangle}$
- **Euclidean space.** For $\mathbb{V} \in \mathbb{C}^n$, the inner product is $\langle u, v\rangle = \sum_{i=1}^n x_i \bar{y_i}$ (where $\bar{y}$ denotes the complex conjugate of y, i.e., $\overline{a + bi} = a - bi$.). For $\mathbb{V} \in \mathbb{R}^n$, this simplifies to $\langle u, v\rangle = \sum_{i=1}^n x_i y_i$.
- **Norm.** The norm associated with the inner product is $\|v\| = \sqrt{\langle v, v\rangle}$. For Euclidean space, this gives us the usual/familiar notion of length of a vector.

For studying the Fourier transform, rather than just finite vectors, we need to consider the space of continuous functions. We can similarly define an inner product for such a space. Let $V = S[a, b]$ be the space of continuous functions $f : [a, b] \to \mathbb{C}$. With the usual definition of sum of functions and multiplication by a scalar, V is a linear space and we can similarly define an inner product

$$\langle f, g\rangle = \int_a^b f(x)\overline{g(x)}dx. \tag{38}$$

Notice how intuitively the continuous functions over the interval are similar to a limiting case where we have an "infinite-length" vector and the discrete summation for the inner product replaced by an integral of the product of the two functions.

We can now introduce orthogonality and orthonormal systems.

- **Orthogonality.** u and v are orthogonal if $\langle u, v\rangle = 0$.
- **Orthogonal system.** An orthogonal system is a sequence of non-zero vectors $\{u_i\}$ such that u_i and u_j are orthogonal $\forall i \neq j$. Note that $\{u_i\}$ can consist of infinite many elements – we can call this an infinite orthogonal system.
- **Orthonormal system.** An orthonormal system is an orthogonal system that additionally satisfies $\|u_i\| = 1, \forall u_i$.

Given an orthonormal system $\{e_i\}$, if $v = \sum_{i=1}^{n} a_i e_i$ then $a_i = \langle v, e_i \rangle$. The proof is straightforward, following just from the linearity of the inner product and orthonormality of the system $\{e_i\}$. When v is in the span of $\{e_i\}$, then $v = \sum_{i=1}^{n} a_i e_i = \sum_{i=1}^{n} \langle v, e_i \rangle e_i$. This may not always be the case, in which case we can define a useful concept:

- **Orthogonal projection.** The orthogonal projection of v on the span of the orthonormal system $\{e_i\}$ can be defined as $\tilde{v} = \sum_{i=1}^{n} \langle v, e_i \rangle e_i$.

With these, we can then define a **closed orthonormal system**. Given an infinite orthonormal system $\{e_i\}$ and inner product space V, we can define it as closed in V if $\forall v \in V$,

$$\lim_{n \to \infty} \left\| v - \sum_{i=1}^{n} \langle v, e_i \rangle e_i \right\| = 0. \tag{39}$$

With a closed infinite orthonormal system, we can then represent any vector v in terms of the system basis $\{e_i\}$. Intuitively, this means that the set $\{e_i\}$ is rich enough to be able to capture all relevant features and express any vector v. In practice, we may only use a finite number of e_i to represent the function, in which case it becomes an approximation rather than being exact. The approximation generally improves as the number of terms taken increases, though in some scenarios the improvement may be slow.

B (**Optional**) Convergence Properties for the Fourier Series

We have not shown that the orthogonal systems Eq. (1) and Eq. (6) are closed, nor that the Fourier series can "reasonably represent" any piecewise continuous periodic function. Details regarding the pointwise, uniform, and norm convergence of the Fourier series are beyond our scope, but we briefly and informally mention some key results here:

- **Pointwise convergence.** Let f be a periodic, piecewise smooth function on $\mathbb{R}$, and let $FS[f]$ be the Fourier series (either using sines/cosines or complex exponentials) for f. Then:
  - $FS[f]$ convergences at every point where f is continuous, i.e., $f(x) = FS[f]|_x$ at every x where f is continuous.
  - If a is a point where the function is discontinuous, then as the number of Fourier terms in the partial sum $N \to \infty$, we get $FS_N[f] \to \frac{1}{2}\left(\lim_{x \to a^-} f(x) + \lim_{x \to a^+} f(x)\right)$.
- **Uniform convergence.** Let f be a continuous, piecewise smooth, periodic function with period p. Then:
  - Its complex exponential Fourier series $FS[f]$ converges uniformly to f. Furthermore, for any real value x and pairs of integers M and N where $M < 0 < N$, we have

$$\left| f(x) - \sum_{k=M}^{N} c_k e^{i 2\pi \omega_k x} \right| \leq \left[\frac{1}{\sqrt{M}} + \frac{1}{\sqrt{N}} \right] B,$$

    where $B = \frac{1}{2\pi} \left(p \int_{-\pi}^{\pi} |f'(x)|^2 dx \right)^{\frac{1}{2}}$, $\forall x$. Thus, the approximation error becomes low as M and N increases as long as none of the derivatives are too large.

– The proof is based on relating coefficients of the Fourier series of f to those of its derivative f', in the same way as what we did for the Fourier transform in Section 4.6.

– We can also get a better bound for smoother functions, i.e., when we know that a function is m-times differentiable, and $f^{(m)}$ is piecewise continuous. Then the bound becomes

$$\left| f(x) - \sum_{k=M}^{N} c_k e^{i2\pi\omega_k x} \right| \leq \left[\frac{1}{\sqrt{M^{2m-1}}} + \frac{1}{\sqrt{N^{2m-1}}} \right] \tilde{B},$$

where $\tilde{B}$ is a constant proportional to the magnitude of $f^{(m)}$.

CS5275 Lecture 8: Information Theory

Jonathan Scarlett

March 17, 2025

Useful references:

- Cover/Thomas book "Elements of Information Theory"
- MacKay book: "Information Theory, Inference, and Learning Algorithms"

Categorization of material: Sections 1–4 are "core material"; the rest is optional.

Note: In the initial sections of these notes, we introduce various information measures and some axiomatic motivation, intuition, and properties. In Sections 5 and 6, we overview compression and communication problems where these information measures naturally arise in their fundamental performance limits.

1 Information of an Event

Getting started.

- If we are told that random event A occurred (e.g., coin came up tails, two dice added up to 7, it rained today), how much "information" have we learned?
- Approach: Quantify information without any regard to *significance* or *importance*. It is only $\Pr[A]$ that matters.
  - Things like "importance" are usually too subjective to quantify.
- Generically speaking, if A occurs with probability p, then

$$\text{Information}(A) = \psi(p)$$

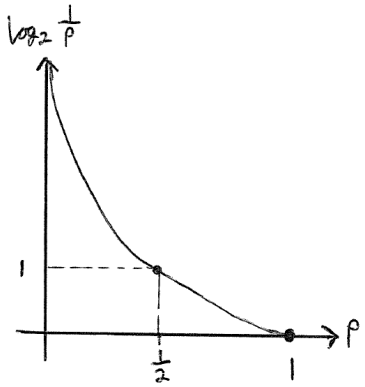
  for some function $\psi(\cdot)$. Perhaps a more intuitive interpretation of $\psi(p)$ is that it quantifies *how surprised we are that event A occurred*. What properties should this function satisfy?

Axiomatic view.

- Here are some very natural properties that we should expect $\psi(p)$ to satisfy:
  1. (Non-negativity) $\psi(p) \geq 0$, i.e., we cannot learn a "negative amount" of information.
  2. (Zero for definite events) $\psi(1) = 0$, i.e., if something was certain to happen, nothing is learned by the fact that it occurred.

3. (Monotonicity) If $p \le p'$, then $\psi(p) \ge \psi(p')$, i.e., the less likely the event was, the more information is learned by the fact that it occurred.
4. (Continuity) $\psi(p)$ is continuous in p, i.e., small changes in probability don't cause drastic changes in information.
5. (Additivity under independence) $\psi(p_1 p_2) = \psi(p_1) + \psi(p_2)$. If A and B are independent events with probabilities p_1 and p_2, then $A \cap B$ has probability $p_1 p_2$, and the information learned from both A and B occurring is the sum of the two individual amounts of information (because they are independent!)

- It can be shown that only $\psi(p) = \log_b \frac{1}{p}$ (for some base $b > 0$) satisfies all three
  - We focus on $b = 2$, which means information is measured in "bits". Another common choice is $b = e$, which means information is measured in "nats".
  - All choices of b are equivalent up to scaling by a universal constant (e.g., number of nats = $(\log_e 2)$ × number of bits). This is much the same as how we can measure distance in meters, kilometers, inches, or miles, but converting from one to another just amounts to scaling.
  - So being told that a probability-p event occurred gives us $\log_2 \frac{1}{p}$ "bits" of information.
  - An illustration:



2 Information of a Random Variable – Entropy

Definition.

- Let X be a discrete random variable with probability mass function (PMF) P_X
- According to the previous section, if we observe $X = x$ then we have learned $\log_2 \frac{1}{P_X(x)}$ bits of information. The **(Shannon) entropy** is simply the average of this value with respect to P_X:

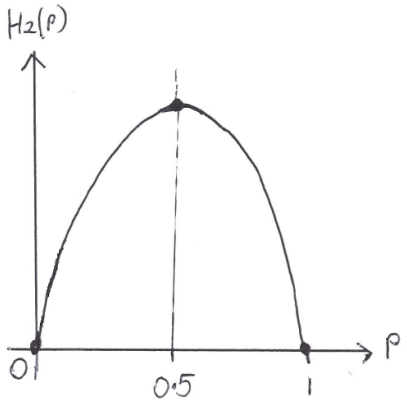
$$\begin{aligned} H(X) &= \mathbb{E}_{X \sim P_X}\left[\log_2 \frac{1}{P_X(X)}\right] \\ &= \sum_x P_X(x) \log_2 \frac{1}{P_X(x)}. \end{aligned}$$

  - Note the convention $0 \log \frac{1}{0} = 0$, which is reasonable since $\lim_{p \to 0} p \log_2 \frac{1}{p} = 0$.
- Can be viewed as a measure of *information in* X or *uncertainty in* X (these are not contradictory)

- **Note.** Here and throughout the vast majority of the course, we only consider *discrete-valued* random variables that can only take on a finite number of values. We will cover *continuous-valued* random variables much later.

Examples.

- Binary source:
  - Suppose $X \sim \text{Bernoulli}(p)$ for some $p \in (0,1)$ (i.e., $P_X(1) = 1 - P_X(0) = p$)
  - Then we get
$$H(X) = p \log_2 \frac{1}{p} + (1-p) \log_2 \frac{1}{1-p}. \tag{1}$$
    The right hand side, as a function of p, is known as the *binary entropy function*. Since this quantity will be used frequently throughout the course, we give it a formal definition: $H_2(p) = p \log_2 \frac{1}{p} + (1-p) \log_2 \frac{1}{1-p}$ for $p \in [0,1]$. An illustration:



- Uniform source:
  - Suppose X is uniform on a finite set $\mathcal{X}$ (i.e., $P_X(x) = \frac{1}{|\mathcal{X}|}$ for each $x \in \mathcal{X}$, where $|\mathcal{X}|$ is the cardinality of $\mathcal{X}$)
  - Then we get
$$H(X) = \mathbb{E}\left[\log_2 \frac{1}{1/|\mathcal{X}|} \right] = \log_2 |\mathcal{X}|.$$
    This is intuitive, e.g., with 10 bits we can produce $|\mathcal{X}| = 2^{10}$ combinations of bits.

(Optional**) Axiomatic view [Shannon].**

- Suppose that X is a discrete random variable taking N values, with probabilities $\mathbf{p} = (p_1, \ldots, p_N)$. If we consider a general information measure of the form
$$\Psi(\mathbf{p}) = \Psi(p_1, \ldots, p_N),$$
  then what properties should it satisfy?

- Three natural properties:
  1. (Continuity) $\Psi(\mathbf{p})$ is continuous as a function of $\mathbf{p}$. Again, small changes in the distribution don't give large changes in information/uncertainty.

2. (Uniform case) If $p_i = \frac{1}{N}$ for $i = 1, \ldots, N$, then $\Psi(\mathbf{p})$ is increasing in N. That is, being uniform over a larger set of outcomes always means more information/uncertainty.
3. (Successive decisions) The following always holds:

$$\Psi(p_1, \ldots, p_N) = \Psi(p_1 + p_2, p_3, \ldots, p_N) + (p_1 + p_2)\Psi\left(\frac{p_1}{p_1 + p_2}, \frac{p_2}{p_1 + p_2}\right).$$

This can be viewed as drawing from the distribution on X by first drawing from the corresponding distribution that doesn't distinguish two symbols (the ones with probabilities p_1 and p_2), and then drawing another random variable to resolve those two symbols if needed (which only happens a fraction $p_1 + p_2$ of the time). The total information/uncertainty is the sum of the information/uncertainty from each of the two stages.

- It can be shown that only $\Psi(X) = \text{constant} \times H(X)$ satisfies all three.

Variations.

- Joint entropy of two random variables (X, Y):

$$\begin{aligned} H(X, Y) &= \mathbb{E}_{(X,Y)\sim P_{XY}}\left[\log_2 \frac{1}{P_{XY}(X, Y)}\right] \\ &= \sum_{x,y} P_{XY}(x, y) \log_2 \frac{1}{P_{XY}(x, y)}. \end{aligned}$$

We can similarly define $H(X, Y, Z)$ or larger collections such as $H(X_1, \ldots, X_n)$.

- Conditional entropy of Y given X:

$$\begin{aligned} H(Y|X) &= \mathbb{E}_{(X,Y)\sim P_{XY}}\left[\log_2 \frac{1}{P_{Y|X}(Y|X)}\right] \\ &= \sum_{x,y} P_{XY}(x, y) \log_2 \frac{1}{P_{Y|X}(y|x)} \\ &= \sum_{x} P_X(x) H(Y|X = x), \end{aligned} \tag{2}$$

where in the last line, $H(Y|X = x) = \sum_y P_{Y|X}(y|x) \log_2 \frac{1}{P_{Y|X}(y|x)}$ is simply the entropy of the distribution $P_{Y|X}(\cdot|x)$ on Y. We can similarly define quantities like $H(Y_1, Y_2|X_1, X_2)$.

  - Intuition: $H(Y|X = x)$ is the uncertainty in Y after having observed that $X = x$. The conditional entropy $H(Y|X)$ simply averages such a quantity over X, so it represents the average remaining uncertainty in Y after observing X.

Example:

- Consider the joint distribution described as follows:

$$P_X(0) = 1 - p, \quad P_X(1) = p,$$

$$P_{Y|X}(y|x) = \begin{cases} 1 - \delta & y = x \\ \delta & y \neq x, \end{cases}$$

where X and Y are both $\{0,1\}$-valued.

- That is, $X \sim \text{Bernoulli}(p)$ and then Y is generated by flipping X with probability δ.
- We have the following:
  - $H(X) = H_2(p)$ as already done above.
  - Similarly, $H(Y) = H_2(q)$ where $q = P_Y(1) = p(1-\delta) + (1-p)\delta$.
  - $H(Y|X) = \sum_x H(Y|X=x)$, but $H(Y|X=x)$ is $H_2(\delta)$ for both x values, so $H(Y|X) = H_2(\delta)$.
  - $H(X;Y)$ could be computed directly based on the 4 joint probabilities $(p\delta, p(1-\delta), (1-p)\delta, (1-p)(1-\delta))$, but an easier way it to use the property $H(X,Y) = H(X) + H(Y|X)$ (see *chain rule* below), giving $H(X,Y) = H_2(p) + H_2(\delta)$.
  - Computing $H(X|Y)$ directly would require using Bayes' rule to get an expression for $P_{X|Y}$ and substituting into $H(X|Y) = \sum_y P_Y(y) H(X|Y=y)$. This gets a bit messy, so is skipped here. (Again, the chain rule gives an easier approach via $H(X,Y) = H(Y) + H(X|Y)$.)

2.1 Properties of Entropy

- **Non-negativity:**

$$H(X) \geq 0$$

with equality if and only if X is deterministic.

  - Intuition: Information/uncertainty cannot be negative
  - Proof: The "information of an event" $\log_2 \frac{1}{p}$ is always non-negative for $p \in [0,1]$, so entropy is the average of a quantity that is always non-negative, and so is itself non-negative. Moreover, only $p = 1$ gives $\log_2 \frac{1}{p} = 0$, so $H(X) = 0$ if and only if X is deterministic.

- **Upper bound:** If X takes values on a finite alphabet $\mathcal{X}$, then

$$H(X) \leq \log_2 |\mathcal{X}|$$

with equality if and only if X is uniform on $\mathcal{X}$. This similarly implies $H(X|Y) \leq \log_2 |\mathcal{X}|$.

  - Intuition: The uniform distribution has the most uncertainty.
  - Proof: Let P be the distribution of X, and let Q be the uniform distribution on $\mathcal{X}$, so that $Q(x) = \frac{1}{|\mathcal{X}|}$ for all x. Then note that

$$\begin{aligned}\sum_x P(x) \log_2 \frac{P(x)}{Q(x)} &= \sum_x P(x) \log_2 \left(|\mathcal{X}| \cdot P(x)\right) \\ &= \log_2 |\mathcal{X}| + \sum_x P(x) \log P(x) \\ &= \log_2 |\mathcal{X}| - H(X).\end{aligned}$$

    In Section 3 we will show that the left-hand side is non-negative for *any* distributions P and Q, with equality if and only if $P = Q$. Specialized to the above choices of P and Q, we get $\log_2 |\mathcal{X}| - H(X) \geq 0$ with equality if and only if P is uniform, as desired.

- **Chain rule (two variables):**

$$H(X,Y) = H(X) + H(Y|X)$$

  - Intuition: The overall information in (X,Y) is the information in X plus the remaining information in Y after observing X.
  - Proof: For $(X,Y) \sim P_{XY}$, we have

$$\begin{aligned} H(X,Y) &= \mathbb{E}\left[\log \frac{1}{P_{XY}(X,Y)}\right] \\ &= \mathbb{E}\left[\log \frac{1}{P_X(X)P_{Y|X}(Y|X)}\right] \\ &= \mathbb{E}\left[\log \frac{1}{P_X(X)} + \log \frac{1}{P_{Y|X}(Y|X)}\right] \\ &= H(X) + H(Y|X). \end{aligned}$$

  - Note: We can swap the roles of X and Y, giving $H(X,Y) = H(Y) + H(X|Y)$.

- Chain rule (general):

$$H(X_1, \ldots, X_n) = \sum_{i=1}^{n} H(X_i|X_1, \ldots, X_{i-1}).$$

  - Intuition: Similar to the two-variable case.
  - Proof: Similar to the two-variable case, but instead use the expansion $P_{X_1 \ldots X_n} = P_{X_1} \times P_{X_2|X_1} \times P_{X_3|X_1X_2} \times \ldots \times P_{X_n|X_1,\ldots,X_{n-1}}$.

- Conditioning reduces[1] entropy:

$$H(X|Y) \leq H(X)$$

  with equality if and only if X and Y are independent.

  - Intuition: Having additional information cannot increase uncertainty *on average*.[2]
  - Proof: Equivalent to the property $I(X;Y) \geq 0$ to be proved in Section 4.1.

- Sub-additivity:

$$H(X_1, \ldots, X_n) \leq \sum_{i=1}^{n} H(X_i)$$

  with equality if and only if $X_1, \ldots, X_n$ are independent.

  - Intuition: The uncertainty in several random variables is no more than the sum of individual uncertainty in each one.
  - Proof: Apply "conditioning reduces entropy" to each summand in the general chain rule formula above.

[1] More precisely, does not increase

[2] In contrast, $H(X|Y=y)$ for a particular y could exceed $H(X)$. For example, suppose that $P_X(0) = 1 - P_X(1) = 0.99$, so that $X = 1$ is very rare. But if Y is sufficiently correlated with X, we could have $P_{X|Y}(0|0) = \frac{1}{2}$, meaning that being told $Y = 0$ made us much less certain about X.

3 A Useful Measure Between Distributions – KL Divergence

- For two PMFs P and Q on a finite alphabet $\mathcal{X}$, the *Kullback-Leibler (KL) divergence* (also known as *relative entropy*) is given by

$$\begin{aligned} D(P\|Q) &= \sum_x P(x) \log_2 \frac{P(x)}{Q(x)} \\ &= \mathbb{E}_{X\sim P}\left[\log_2 \frac{P(X)}{Q(X)}\right]. \end{aligned}$$

- Can be viewed as a kind of "distance" between P and Q, but it is not a distance function in the mathematical sense (in general it is not symmetric and doesn't satisfy the triangle inequality).

- **Claim.** For any distributions P and Q, we have

$$D(P\|Q) \geq 0$$

with equality if and only if $P = Q$.

  - Proof:

$$\begin{aligned} -D(P\|Q) &= \sum_x P(x) \log \frac{Q(x)}{P(x)} \\ &\overset{(a)}{\leq} \sum_x P(x)\left(\frac{Q(x)}{P(x)} - 1\right) \\ &= \sum_x Q(x) - \sum_x P(x) \\ &= 0, \end{aligned}$$

    where (a) uses the inequality $\log \alpha \leq \alpha - 1$, which is easily verified graphically. Equality holds in $\log \alpha \leq \alpha - 1$ if and only if $\alpha = 1$, which means that equality holds in (a) if and only if $\frac{Q(x)}{P(x)} = 1$ for all x (i.e., $P = Q$).

- **(**Optional**)** While we will focus more on entropy and mutual information, we note that KL divergence has other useful properties beyond the one above, e.g.:

  - A chain rule holds; for two variables, it is

$$D(P_{XY}\|Q_{XY}) = D(P_X\|Q_X) + D(P_{Y|X}\|Q_{Y|X}|P_X),$$

    where $D(P_{Y|X}\|Q_{Y|X}|P_X) = \sum_x P_X(x) D(P_{Y|X}(\cdot|x)\|Q_{Y|X}(\cdot|x))$.

  - A data processing inequality holds: Given P_X, Q_X, if we form P_Y, Q_Y by applying the same transformation to X (i.e., $P_Y(y) = \sum_x P_X(x)V(y|x)$ and $Q_Y(y) = \sum_x Q_X(x)V(y|x)$ for some randomized transformation V), then

$$D(P_Y\|Q_Y) \leq D(P_X\|Q_X).$$

In other words, "processing" X to produce Y can only make the distributions become closer together, not further apart.

- The KL divergence (and in fact, also entropy and mutual information) is used extensively in other fields like statistics and machine learning. Some example uses (stated only very roughly here) are:
  - In data compression, if the true source is distribution is P but we use an algorithm that wrongly assumes it is Q, then we pay a penalty of $D(P\|Q)$ in the average number of bits per symbol;
  - In statistics, if $\mathbf{X} = (X_1, \ldots, X_n)$ is i.i.d. with $X_i \sim Q$, then the probability that the observed proportions of symbols match P is roughly $2^{-nD(P\|Q)}$ when n is large. You can look up *Sanov's theorem* for a more precise and more general statement.

4 Information Between Random Variables – Mutual Information

Definition.

- Mutual information:

$$I(X;Y) = H(Y) - H(Y|X).$$

- Intuition:
  - $H(Y)$ is the *a priori uncertainty* in Y
  - $H(Y|X)$ is the *remaining uncertainty* in Y after observing X (on average)
  - Hence, $I(X;Y)$ is the *amount of information about Y we resolve by observing X* (on average).

Variations.

- Joint version:

$$I(X_1, X_2; Y_1, Y_2) = H(Y_1, Y_2) - H(Y_1, Y_2|X_1, X_2).$$

- Conditional version:

$$I(X;Y|Z) = H(Y|Z) - H(Y|X, Z).$$

Examples.

1. If X and Y are independent, then it is straightforward to compute $H(Y|X) = H(Y)$, giving $I(X;Y) = 0$ (i.e., independent random variables do not reveal any information about each other).
2. If $Y = X$, then it is straightforward to compute $H(Y|X) = H(X|X) = 0$, and hence $I(X;X) = H(X)$ (i.e., the amount of information a random variable reveals about itself is the entropy).
3. In the example given shortly after Eq. (2), we computed $H(Y|X) = H_2(\delta)$ and $H(Y) = H_2(p(1-\delta) + (1-p)\delta)$, which we can substitute into $I(X;Y) = H(Y) - H(Y|X)$ to get the mutual information. In particular, when $p = \frac{1}{2}$ we simply get $I(X;Y) = 1 - H_2(\delta)$.

4.1 Properties of Mutual Information

- **Alternative forms:**

$$\begin{aligned} I(X;Y) &= D(P_{XY} \| P_X \times P_Y) \\ &= \mathbb{E}\left[\log_2 \frac{P_{XY}(X,Y)}{P_X(X)P_Y(Y)} \right] = \sum_{x,y} P_{XY}(x,y) \log_2 \frac{P_{XY}(x,y)}{P_X(x)P_Y(y)} \\ &= \mathbb{E}\left[\log_2 \frac{P_{Y|X}(Y|X)}{P_Y(Y)} \right] = \sum_{x,y} P_{XY}(x,y) \log_2 \frac{P_{Y|X}(y|x)}{P_Y(y)}. \end{aligned}$$

  - Proof: Substituting $H(Y) = \mathbb{E}\big[\log_2 \frac{1}{P_Y(Y)}\big]$ and $H(Y|X) = \mathbb{E}\big[\log_2 \frac{1}{P_{Y|X}(Y|X)}\big]$ into the definition of mutual information gives $I(X;Y) = \mathbb{E}\big[\log_2 \frac{P_{Y|X}(Y|X)}{P_Y(Y)}\big]$. Multiplying the numerator & denominator by $P_X(X)$ gives $\mathbb{E}\big[\log_2 \frac{P_{XY}(X,Y)}{P_X(X)P_Y(Y)}\big]$, from which the remaining equalities follow easily.
  - The expression $D(P_{XY} \| P_X \times P_Y)$ has the interpretation of measuring "how far X and Y are from being independent".

- **Symmetry:** We have

$$I(X;Y) = H(X) + H(Y) - H(X,Y)$$

  and in particular

$$I(X;Y) = I(Y;X)$$

  which also implies

$$I(X;Y) = H(X) - H(X|Y).$$

  - Intuition: X and Y reveal an equal amount of information about each other (or maybe this is not that intuitive!)
  - Proof: We have from the above alternative form that

$$\begin{aligned} I(X;Y) &= \mathbb{E}\left[\log_2 \frac{P_{XY}(X,Y)}{P_X(X)P_Y(Y)} \right] \\ &= \mathbb{E}\left[\log_2 \frac{1}{P_X(X)} + \log_2 \frac{1}{P_Y(Y)} + \log_2 P_{XY}(X,Y) \right] \\ &= H(X) + H(Y) - H(X,Y), \end{aligned}$$

    where we first expanded the logarithm, and then applied the definition of (joint) entropy.

- **Non-negativity:** $I(X;Y) \geq 0$ with equality if and only if X and Y are independent.

  - Intuition: One random variable cannot tell us a "negative amount" of information about the other.
  - Proof: Using the above-established identity $I(X;Y) = D(P_{XY} \| P_X \times P_Y)$, this is just a special case of $D(P\|Q) \geq 0$ with equality if and only if $P = Q$.

- **Upper bounds:** We have

$$\begin{aligned} I(X;Y) &\leq H(X) \leq \log_2 |\mathcal{X}| \\ I(X;Y) &\leq H(Y) \leq \log_2 |\mathcal{Y}|. \end{aligned}$$

- Intuition: The information X reveals about Y (mutual information) is at most the prior information in X (entropy).
- Proof: To show that $I(X;Y) \leq H(X)$, combine $I(X;Y) = H(X) - H(X|Y)$ (see above) and $H(X|Y) \geq 0$ (conditional or unconditional entropy is never negative). We already showed $H(X) \leq \log_2 |\mathcal{X}|$ earlier, and the remaining claims follow by symmetry, reversing the roles of X and Y.

- **Chain rule:**
$$I(X_1, \ldots, X_n; Y) = \sum_{i=1}^{n} I(X_i; Y | X_1, \ldots, X_{i-1}).$$
  - Intuition: Similar to the chain rule for entropy.
  - Proof: Write $I(X_1, \ldots, X_n; Y) = H(X_1, \ldots, X_n) - H(X_1, \ldots, X_n|Y)$ and apply the chain rule for entropy to both terms.

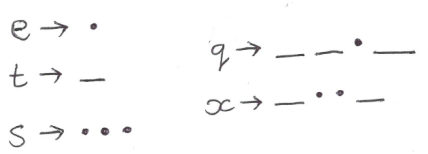
- **Data processing inequality:** If Z depends on (X, Y) only through Y (often stated via the terminology "$X \to Y \to Z$ forms a Markov chain", and equivalent to the statement "X and Z are conditionally independent given Y"), then
$$I(X; Z) \leq I(X; Y).$$
  - Intuition: Processing Y (to produce Z) cannot increase the information available regarding X.
  - Proof: As stated above, the statement "Z depends on (X, Y) only through Y" is equivalent to "Z and X are conditionally independent given Y". This means that the property $P_{Z|XY} = P_{Z|Y}$ (as assumed in the result) is equivalent to $P_{X|YZ} = P_{X|Y}$. To deduce the result, we write
$$\begin{aligned} I(X;Z) &\overset{(a)}{=} H(X) - H(X|Z) && (3)\\ &\overset{(b)}{\leq} H(X) - H(X|Y,Z) && (4)\\ &\overset{(c)}{=} H(X) - H(X|Y) && (5)\\ &\overset{(d)}{=} I(X;Y), && (6) \end{aligned}$$
    where (a) and (d) use the definition of mutual information, (b) follows since conditioning reduces entropy, and (c) holds because $H(X|Y,Z) = \mathbb{E}\big[\log \frac{1}{P_{X|YZ}(X|Y,Z)}\big] = \mathbb{E}\big[\log \frac{1}{P_{X|Y}(X|Y)}\big] = H(X|Y)$ by the above-established fact $P_{X|YZ} = P_{X|Y}$.
  - Variations:
    * If $X \to Y \to Z$ then $I(X;Z) \leq I(Y;Z)$.
    * If $W \to X \to Y \to Z$ then $I(W;Z) \leq I(X;Y)$.

- **Partial sub-additivity:** If $(Y_1, \ldots, Y_n)$ are conditionally independent given $(X_1, \ldots, X_n)$, and in addition Y_i depends on $(X_1, \ldots, X_n)$ only through X_i, then
$$I(X_1, \ldots, X_n; Y_1, \ldots, Y_n) \leq \sum_{i=1}^{n} I(X_i; Y_i).$$
  (You can try proving this as an exercise, or see Notes #1 at https://www.comp.nus.edu.sg/~scarlett/CS3236_notes/ for the proof.) However, without the conditional independence assumptions, this property may fail to hold.

5 (**Optional**) A Brief Overview of Data Compression

Familiar examples of compression.

- When we compress a file to .zip or .rar it gets smaller, and yet we can still recover the contents. How/why is this possible? This is the problem of **lossless compression**.
- When we convert a file from .bmp to .jpeg, we lose some quality, but hopefully not too much. However, we cannot convert back to the higher-quality image. This is the problem of **lossy compression**.
- Concepts in compression go further back than computers – recall Morse code:



Example 1: Sparse binary string.

- Suppose that we want to efficiently store

$$0000000\mathbf{1}000000000000000000000\mathbf{1}000000000000000000000000\mathbf{1}000000000,$$

  i.e., a string of length 64 with only three 1s and the rest 0s.
- Storing this "as is" requires 64 bits (a bit being a 1 or 0).
- Alternative scheme:
  - Index the string positions from 0 to 63, and consider their binary format (e.g., $0 \to 000000$, $7 \to 000111$, $63 \to 111111$)
  - Store the 3 positions where the long string has value 1, using 6 bits per position.
- This permits only 18 bits of storage instead of 64.

Example 2: Equal number of 1s and 0s.

- Suppose that we need a system that can compress sequences of the form

$$1011010101100111010011001100100101101011101010100010010010100 11,$$

  i.e., still length 64, but now half ones and half zeros.
- Again, storing "as is" requires 64 bits.
- The number of strings with half zeros and half ones is $\binom{64}{32}$. Let's aim to compress them down to some number $L < 64$ of bits. How small can L be?
- With L bits (each 0 or 1), we can make 2^L combinations. Since each of the $\binom{64}{32}$ strings have to be stored as a different combination, we clearly need $2^L \geq \binom{64}{32}$, or equivalently

$$L \geq \log_2 \binom{64}{32} \approx 60.7.$$

(More generally, compressing N different strings down to L bits without loss requires $L \geq \log_2 N$.)

- So we can't hope to do much better than direct storage!

Example 3: English text.

- English text clearly has a fair bit of redundancy, since we can "throw away" several letters but still (usually) recover the original text:

C_N Y_ _ F_LL _N TH_ V_W_LS _N TH_S S_NT_NC_?

- If we (somewhat naively) store English text in some binary format in a letter-by-letter fashion, we can exploit the fact that some letters are more common than others, e.g., map 'e' to a short binary sequence, and 'x' to a long binary sequence.
  - Morse code is an early example of this idea (but not quite "binary"!)
  - Can we construct an "optimal" mapping?
- The savings are much greater if we exploit the fact that different *groups* of letters are more likely to appear together (e.g., if we have already seen "Fill in the blan", then there is clearly a much more likely letter than 'e' coming next!)
- Spoiler: While it requires 5 bits (or at least $\log_2 27 \approx 4.75$) to uniquely identify one of 27 characters ('a' to 'z' and also spaces), the actual "information content" of each letter in English text is only about 1.34 bits. This will mean that we can compress down by a factor of at least $3\times$ without losing anything.
  - See the appendix of this document for further details

Information-theoretic viewpoint.

- Information theory adopts probabilistic models, e.g., a string of 1000 English characters is modeled by some joint distribution $P_{X_1 X_2 \ldots X_{999} X_{1000}}$, where each X_i takes some value in $a \ldots z$ (or space).
  - Probabilistic modeling can provide a very good trade-off between accuracy in modeling the real world vs. tractability of the mathematical analysis.
- Two distinct approaches to compression:
  - (Variable-length) Map more probable sequences to shorter binary strings, at the expense of mapping less probable sequences to longer strings. **How low can the average length be?**
  - (Fixed-length) Map the most probable sequences to binary strings of a given length, at the expense of not having enough such strings for the low-probability sequences. **How low can the length be while having a very low probability of failure?**
- **Source coding theorem (informal).** In both of these settings, the fundamental compression limit is given by the Shannon entropy H. The (average) storage length can be arbitrarily close to H, but can never be any lower than H.

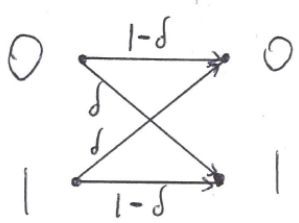
6 (**Optional**) A Brief Overview of Data Communication

Familiar examples of communication.

- When military pilots want to read a sequence of letters over an intercom, they use "alpha", "bravo", "Charlie", etc.
- If someone on the other end of the phone is having trouble hearing us, we might repeat the same thing 2–3 times to make sure they hear it.
- If we're talking to someone in their non-native language, we might talk slower.
- Our WiFi slows down as we move further away from the router.
- Common theme: Send information *more slowly* but also *more reliably.*
  - For a given reliability, how slow do we need to go?

Simple communication setting.

- Let's suppose that we are communicating in binary:
  - A "transmitter" sends a sequence of 0s and 1s
  - A "receiver" receives the sequence *with some corruptions*: Each bit is flipped (from 0 to 1, or from 1 to 0) independently with probability $\delta \in \left(0, \frac{1}{2}\right)$.
  - This is depicted in the following "channel transition diagram":



  - e.g., The sequence 01101000 might be received as 0**0**101**1**00 (two corruptions)

Approach 1: Uncoded communication.

- Suppose that the transmitter wants to send one of 16 messages (e.g., it has done a weather reading and wants to send one of 16 possibilities among "sunny", "rainy", "partly cloudy", etc.)
- Naively, it can do this by mapping each outcome to a unique sequence of 4 bits (e.g., sunny $\to$ 0000, rainy $\to$ 1010, etc.)
- Since each bit is flipped with probability δ, the probability of all 4 bits coming out correct is $(1-\delta)^4$. For instance, if $\delta = 0.1$, we have $\mathbb{P}[\text{correct}] = 0.9^4 = 0.6561$.
- Things get worse as we send more messages, e.g., if we encode one of $2^8 = 256$ messages to a length-8 binary string and transmit it, we get $\mathbb{P}[\text{correct}] = (1-\delta)^8$, which is roughly 0.43 when $\delta = 0.1$.

Approach 2: Repetition code.

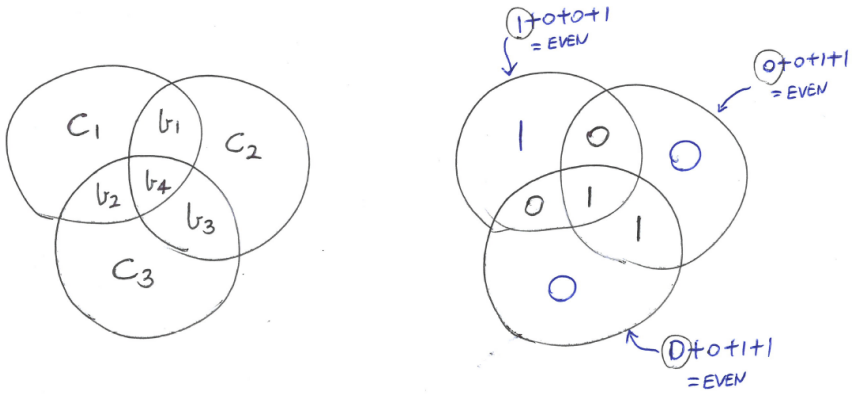
- As mentioned above, let's try transmitting slower but more reliably!
- Let's start with just sending one of two messages, which we will label as 0 and 1.
- Repetition code R_3 of length 3:
  - To send "0", transmit the sequence "000"
  - To send "1", transmit the sequence "111"
  - At the receiver, take the majority vote (e.g., 000 or 010 get decoded as "0", whereas 111 or 110 get decoded as "1")
- Clearly, we get correct decoding if there are no flips or one flip, so $\mathbb{P}[\text{correct}] = (1-\delta)^3 + 3\delta(1-\delta)^2$, which equals 0.972 when $\delta = 0.1$.
- We can then transmit, say, one of 16 messages by mapping (e.g.) 0101 to 000111000111. The probability of getting back the correct message is $0.972^4 \approx 0.893$
  - A fair bit more reliable than uncoded – but three times slower!
- We can do the same with more repetitions:
  - e.g., map 0101 to 000000011111110000000011111111 (repetition code R_7)
  - Any given bit (out of the 4 sent) is decoded correctly with probability

$$(1-\delta)^7 + 7\delta(1-\delta)^6 + \binom{7}{2}\delta^2(1-\delta)^5 + \binom{7}{3}\delta^3(1-\delta)^4 \approx 0.9973$$

    (This is the probability that a Binomial$(7, 0.1)$ random variable is at most 3)
  - The overall message is decoded correctly with probability $\approx 0.9973^4 \approx 0.989$.
  - Now the communication is very reliable, but we are 7 times slower than uncoded! Do we have to keep getting slower and slower?

Approach 3: Hamming code.

- Here we give a famous example of how to map a binary string of length 4 (so 16 messages) to a binary string of length 7 while still being able to correct one bit flip.
- The technique: In the following figure, fill in $b_1b_2b_3b_4$ ('b' for 'bit') with the original four bits, and assign $c_1c_2c_3$ ('c' for 'check') the values that make the three bits in their circle add up to an even number. (An example is shown on the right)



- Observe that any single bit flip (whether it be one of the b_i or one of the c_i) changes a unique combination of circles from "even" to "odd"! Therefore, if a single bit flip occurs, we can uniquely identify which bit caused it, and therefore correct it.
  - We can also distinguish the case that no bit flips occurred, and hence all 3 circles remain "even".
- Therefore

$$\begin{aligned}\mathbb{P}[\text{correct}] &\geq \mathbb{P}[\text{zero or one bit flip(s)}] \\ &= (1-\delta)^7 + 7\delta(1-\delta)^6 \\ &\approx 0.85,\end{aligned}$$

  where the last line holds when $\delta = 0.1$.
- Nearly as reliable as the repetition code, despite mapping to only 7 bits instead of 12! (i.e., we are transmitting a fair bit "less slowly")

Information-theoretic results.

- **Definition.** If we map k bits to n bits in the encoding procedure, then the *rate* is $\frac{k}{n}$ (e.g., $\frac{4}{7}$ for the above Hamming code, $\frac{1}{3}$ for the repetition code, 1 for uncoded)
- Clearly, there is an inherent trade-off between rate and error probability.
  - Higher rate = Send faster
  - Lower error probability = Send more reliably
- **Channel coding theorem (informal).** There exists a channel-dependent quantity called the *(Shannon) capacity* C such that arbitrarily small error probability can be achieved for all rates less than C, but for no rates higher than C. Specifically, we have $C = \max_{P_X} I(X;Y)$, where P_X is optimized and $P_{Y|X}$ is the channel transition law.
  - In the above example with $\delta = 0.1$, we get $C \approx 0.531$ (more generally $C = 1 - H_2(\delta)$). So for arbitrarily small error probability (e.g., $\mathbb{P}[\text{error}] \leq 10^{-10}$), not only is it unnecessary to multiply the number of bits by a higher and higher number, but we can get away with fewer than double the original number (!)

– Caveat: We may need to code over a much longer block length (e.g., map $k = 5000$ bits to $n = 10000$ bits, rather than mapping $k = 3$ bits to $n = 6$ bits)

Principles of information theory:

- First fundamental limits, then practical methods
- First asymptotic results, then finite-length refinements
- Mathematically tractable yet powerful probabilistic models

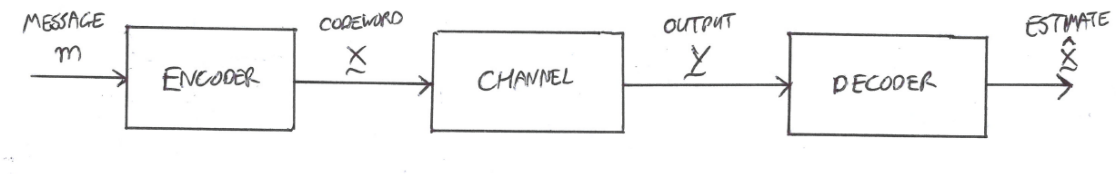
7 (**Optional**) Proof Outline for the Channel Coding Theorem

- The proofs of the source and channel coding theorems use similar ideas; we will focus on the latter.
- The proof is split into two statements:
  - (Achievability) For any transmission rate $R < C$, there exists a sequence of codes (indexed by the block length n) such that $P_e \to 0$.
  - (Covnerse) For any transmssion rate $R > C$, it is impossible to obtain $P_e \to 0$, regardless of the choice of code (in fact a stronger statement $P_e \to 1$ can be shown).

  Note that 'Achievability' means 'mathematical existence property' and 'Converse' means 'mathematical impossibility result'.

Outline of achievability proof:

- The setup is depicted as follows:



- We need to specify the behavior of both the encoder and decoder:
  - (Encoder) Given a message m, produce a length-n codeword $\mathbf{x}^{(m)}$. The codewords can be collected into a codebook $\{\mathbf{x}^{(i)}\}$.
  - (Decoder) Given knowledge of the coedbook but not the specific message m, and also given the received sequence $\mathbf{y}$, produce an estimate $\hat{m}$.
- Codebook design:
  - Designing a good codebook explicitly is very difficult (it was the focus of decades of research after information theory was introduced!)
  - Instead, the original proofs show the *existence* of good codebooks in a non-constructive way.

- This was done using *random coding* (a case of *the probabilistic method*) – analyze the average performance of a codebook whose codewords have entries drawn independently from P_X. If the average performance is "good", then the best specific code's performance is certainly no worse.

- Decoder design and analysis:
  - There are multiple decoding rules that suffice to prove the channel coding theorem, with the most powerful (optimal) rule being maximum-likelihood decoding. The rule usually found in textbooks is *joint typicality decoding*.
  - Roughly, joint typicality decoding is based on searching for a codeword that "looks like" it was drawn i.i.d. from $P_X \times P_{Y|X}$ (with P_X from random coding, and $P_{Y|X}$ being the channel).
  - Under random coding, it can be shown that the correct codeword passes this joint typicality condition with probability approaching one, whereas any incorrect codeword only passes it with probability roughly $2^{-nI(X;Y)}$. Hence, if the number of messages is 2^{nR} with $R < I(X;Y)$, then the decoder succeeds with high probability.
  - Optimizing P_X gives the capacity formula $C = \max_{P_X} I(X;Y)$.

Outline of converse proof:

- The proof is roughly outlined as follows:
  - Relate the error probability to $I(m;\hat{m})$, where m a uniformly random message and $\hat{m}$ is the final estimate. Intuitively, if $\hat{m} = m$ with high probability then $I(m;\hat{m})$ should be large, and the contrapositive version of this leads to a statement on the error probability.
  - Use a property called *data processing inequality* to bound $I(m;\hat{m}) \leq I(\mathbf{X};\mathbf{Y})$, where $\mathbf{X}$ is the transmitted codeword and $\mathbf{Y}$ is the channel output. Intuitively, the end-to-end information that $\hat{m}$ reveals about m cannot exceed the bottleneck imposed by $(\mathbf{X},\mathbf{Y})$ in between.
  - Use mutual information properties (e.g., chain rule) and the memoryless channel assumption to show that $I(\mathbf{X};\mathbf{Y}) \leq \sum_{i=1}^{n} I(X_i;Y_i)$. This is upper bounded by nC since $C = \max_{P_X} I(X;Y)$, and putting everything together completes the proof.

- The first of these steps uses *Fano's inequality*, which is very widespread for proving converse results in information theory and statistics. In generic notation (renaming $(m,\hat{m})$ to $(X,\hat{X})$), the inequality is

$$H(X|\hat{X}) \leq H_2(P_\mathrm{e}) + P_\mathrm{e} \log_2\big(|\mathcal{X}| - 1\big),$$

  where $P_\mathrm{e} = \mathbb{P}[\hat{X} \neq X]$, and $\mathcal{X}$ is the set of values X can take.
  - Intuition. To resolve the uncertainty in X given $\hat{X}$, we can first ask whether the two are equal, which bears uncertainty $H_2(P_\mathrm{e})$. In the case that they differ, which only occurs a fraction P_e of the time, the remaining uncertainty is at most $\log_2\big(|\mathcal{X}| - 1\big)$, since the uniform distribution maximizes entropy.
  - Notice that mutual information doesn't appear above, but when X is uniform we have $I(X;\hat{X}) = H(X) - H(X|\hat{X})$, and substituting Fano's inequality (with $\log_2\big(|\mathcal{X}| - 1\big)$ weakened to $\log_2|\mathcal{X}|$ and $H_2(P_\mathrm{e})$ weakened to 1) and $H(X) = \log_2|\mathcal{X}|$ gives the following:

$$I(X;\hat{X}) \geq (1 - P_\mathrm{e}) \log_2|\mathcal{X}| - 1,$$

or equivalently $P_e \geq 1 - \frac{I(m;\hat{m})+1}{\log_2 |\mathcal{X}|}$. (Intuitively, to have small P_e we need the "learned information" $I(X;\hat{X})$ to match the prior uncertainty $\log_2 |\mathcal{X}|$.)

8 (**Optional**) Broader Uses of Information Theory

Information theory is used extensively in computer science, statistics, machine learning, and beyond, e.g.:

- Lower bounds on sample/query complexity for statistical problems (e.g., estimation, optimization, randomized algorithms)
- Information-theoretic analysis of algorithms, both computationally efficient and inefficient (e.g., an interesting recent example is watermarking of Large Language Models)
- Information measures used in machine learning and other areas (e.g., in sequential decision-making, make decisions that maximize information gain)

See Notes #7 at `https://www.comp.nus.edu.sg/~scarlett/CS3236_notes` for a more detailed overview. Below we give one example in the first category.

Example: Binary search

- Suppose that there are n integers sorted in non-decreasing order, and we want to find the first one to exceed a threshold γ. We can only interact with the list by asking questions of the form "Does the i-th element exceed γ?".
  - We will focus on the case that the allowed number of queries Q is pre-specified (with a value that may depend on n), but this can easily be relaxed.
- Noiseless case: (Every query answer is correct)
  - We can solve this using binary search: Query the middle element, then recurse left or right depending on the answer, and after roughly $Q \approx \log_2 n$ iterations we will have the answer.
  - A simple counting argument shows that no algorithm can do better: If the list is of the form $0 \dots 01 \dots 1$ and $\gamma = \frac{1}{2}$, then there are n possible locations of the transition from 0 to 1. But the algorithm must produce a different output for each such case, so the list of query answers must also be different. With Q queries there are 2^Q possible sequences of outcomes, so we need $2^Q \geq n$, or $Q \geq \log_2 n$.
  - Fano's inequality (see below) can also used to deduce a similar conclusion, with the mutual information simply bounded upper by the number of queries (because each query outcome is binary so can only contribute at most 1 bit).
  - Similar ideas can be applied to other problems, e.g., for comparison-based sorting we get that $\log_2(n!) = \Theta(n \log n)$ comparisons are needed.
- Noisy case: (Each query answer is only correct with probability $1 - \delta$)
  - Having each query answer be flipped w.p. δ amounts to having the answers passed through a binary symmetric channel (BSC), so we can view this through the lens of communication.

– In particular, if $\mathbf{X}$ is the set of queries made and $\mathbf{Y}$ is the sequence of responses, then an argument based on Fano's inequality gives the following lower bound on error probability:

$$P_{\mathrm{e}} \geq 1 - \frac{I(\mathbf{X};\mathbf{Y}) + 1}{\log_2 n}$$

when we consider sequences of the form $0 \ldots 01 \ldots 1$ with the transition from 0 to 1 being uniformly random over n possibilities.

– But by the same analysis as in channel coding, we can derive $I(\mathbf{X};\mathbf{Y}) \leq QC$ where C is the BSC capacity and Q is the number of queries.

– It follows that Q needs to be roughly $\frac{\log_2 n}{C}$ to have high reliability (i.e., $P_{\mathrm{e}} \approx 0$). Using the channel capacity formula $C = 1 - H_2(\delta)$, this can further be shown to behave as $\Theta\big(\frac{\log_2 n}{(1/2-\delta)^2}\big)$.

– Algorithms matching this query complexity to within constant factors is given in the paper "Computing with noisy information" (1994), and the query complexity with precise constants is given in the paper "Optimal bounds for noisy sorting" (2023).

(**Optional**) Appendix: Entropy of English Text

- Shannon's famous 1948 paper discussed several (intentionally over-simplified) probabilistic models for generating English text; see Figure 1 below.
- Stated differently, #3 generates each letter conditioned on the previous one, #4 conditions on the previous two, #5 lets the "alphabet" $\mathcal{X}$ be the set of all words rather than the set of all characters and generates each word independently, and #6 generates each word conditioned on the previous one.
- **Fundamental question:** How much information (entropy) does each letter of English text tell us?
  – The entropy $H(X)$ of a single character doesn't capture the fact that previous characters help in predicting the next one.
  – As detailed in Chapter 4 of Cover/Thomas, a more meaningful measure in such scenarios is

$$H(X_n | X_1, \ldots, X_{n-1}),$$

representing the uncertainty of a given character given all of the previous ones.

  – Fitting a model to English text and then calculating the entropy of that model is prone to be inaccurate (too complex to fit a very accurate model!) – is there a simpler approach?
- **Key idea:** The entropy is closely related to *how many guesses are needed (on average) before the correct character is guessed*, by an optimal guessing algorithm.
  – Intuitively, entropy is a measure of uncertainty, and higher uncertainty means more guesses will be needed on average.
  – Writing an optimal guessing algorithm is hard (though an interesting machine learning problem!), so experiments were done under the assumption that *humans are near-optimal guessers*.

1. Zero-order approximation (symbols independent and equiprobable).

   XFOML RXKHRJFFJUJ ZLPWCFWKCYJ FFJEYVKCQSGHYD QPAAMKBZAACIBZL-HJQD.

2. First-order approximation (symbols independent but with frequencies of English text).

   OCRO HLI RGWR NMIELWIS EU LL NBNESEBYA TH EEI ALHENHTTPA OOBTTVA NAH BRL.

3. Second-order approximation (digram structure as in English).

   ON IE ANTSOUTINYS ARE T INCTORE ST BE S DEAMY ACHIN D ILONASIVE TU-COOWE AT TEASONARE FUSO TIZIN ANDY TOBE SEACE CTISBE.

4. Third-order approximation (trigram structure as in English).

   IN NO IST LAT WHEY CRATICT FROURE BIRS GROCID PONDENOME OF DEMONS-TURES OF THE REPTAGIN IS REGOACTIONA OF CRE.

5. First-order word approximation. Rather than continue with tetragram, ... , n-gram structure it is easier and better to jump at this point to word units. Here words are chosen independently but with their appropriate frequencies.

   REPRESENTING AND SPEEDILY IS AN GOOD APT OR COME CAN DIFFERENT NAT-URAL HERE HE THE A IN CAME THE TO OF TO EXPERT GRAY COME TO FURNISHES THE LINE MESSAGE HAD BE THESE.

6. Second-order word approximation. The word transition probabilities are correct but no further struc-ture is included.

   THE HEAD AND IN FRONTAL ATTACK ON AN ENGLISH WRITER THAT THE CHAR-ACTER OF THIS POINT IS THEREFORE ANOTHER METHOD FOR THE LETTERS THAT THE TIME OF WHO EVER TOLD THE PROBLEM FOR AN UNEXPECTED.

Figure 1: Excerpt from Shannon's paper.

  - Using some theory behind the "optimal guessing" viewpoint, and observing the average number of guesses that several humans required, it was estimated that the entropy of English text is at most **1.34 bits per character**
  - This is much less than the highest possible value of $\log_2 27 \approx 4.75$ with 27 characters (a-z and "space")
  - Note: Recent developments in Language Models suggest that the "correct" value may even be significantly smaller than 1.34 (details omitted).

- See Chapter 6 of Cover/Thomas for further details.

CS5275 Lecture 9: Error-Correcting Codes

Jonathan Scarlett

March 24, 2025

Useful references:

- Blog post on Hamming codes[1]
- TCS Toolkit lectures: https://www.youtube.com/watch?v=H7XFslRXJys and the 4 videos after
- Cover/Thomas "Elements of Information Theory", Section 7.11
- MacKay "Information Theory, Inference, and Learning Algorithms", Sections 11.4–11.5, Chapters 13–14
- (Beyond the scope of this course) Survey article on coding[2] and/or the advanced textbook "Modern Coding Theory" (Richardson and Urbanke)

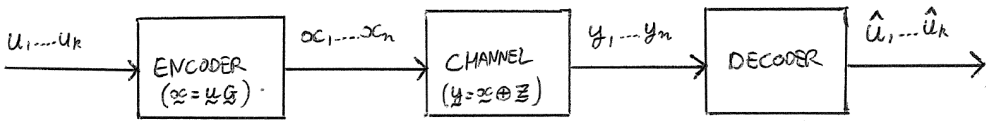
Categorization of material:

- Core material: Sections 1–5 except when labeled "optional"
- Extra material: Section 6 (Reed-Solomon)

(Exam will strongly focus on "Core". Take-home assessments may occasionally require consulting "Extra".)

1 Background: Reliable Communication

- The communication problem was already mentioned in the optional part of the information theory lecture (optional part), and is slightly modified here to highlight that what we are sending is *bits*.
- The setup is summarized as follows (we will explain the equation $\mathbf{x} = \mathbf{uG}$ later):



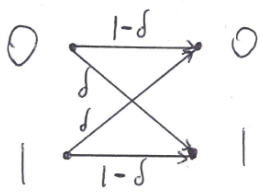
  - $\mathbf{u} = (u_1, \dots, u_k)$ is a sequence of k bits that we would like to send.
  - That sequence gets encoded into a binary codeword $\mathbf{x} = (x_1, \dots, x_n)$ (later we will also consider non-binary codewords), where $n \geq k$ because the idea of encoding is to *add redundancy that allows resilience to bits getting flipped*.

[1] https://jeremykun.com/2015/03/02/hammings-code/
[2] https://arxiv.org/pdf/1908.09903.pdf

- The codeword $\mathbf{x} = (x_1, \ldots, x_n)$ is sent through a communication channel to produce $\mathbf{y} = (y_1, \ldots, y_n)$, which is also binary. The channel can be described as $\mathbf{y} = \mathbf{x} \oplus \mathbf{z}$, where $\mathbf{z} \in \{0,1\}^n$ indicates which bits are flipped, and the operator $\oplus$ is XOR applied bit-by-bit.
- These output bits are used to construct an estimate $\hat{\mathbf{u}} = (\hat{u}_1, \ldots, \hat{u}_k)$ of the original k bits.

- Our goal is to achieve *speed* and *reliability* in communication:
  - *Reliability* means ensuring that $\hat{\mathbf{u}} = \mathbf{u}$ (or, if the channel is modeled as being random, having a *high probability* of this occurring).
  - *Speed* is captured by the notion of *rate*:
$$R = \frac{k}{n}.$$
For example, if the encoder maps k bits to $n = 3k$ bits, then the rate is $\frac{1}{3}$.

- We will consider two types of modeling for the channel:
  - Under a probabilistic model, each y_i is assumed to be produced from the corresponding x_i by passing it through a *binary symmetric channel* (BSC):



That is, the output is correct with probability $1 - \delta$, but flipped with probability δ.
  - An alternative approach is to have no probabilistic model, but simply *assume that at most δn bits are flipped* for some $\delta \in (0, 1)$.

- In the probabilistic case, much more general channel models are possible, with transition law $P_{Y|X}$ and possibly non-binary alphabets, but we will focus on the BSC.

- Another model of particular interest is the *erasure model*, where (in the probabilistic case) $Y = X$ with probability $1 - \epsilon$, but Y equals an "erasure symbol" with probability ϵ, with that symbol indicating "x_i is unknown". Similarly, in the non-probabilistic case, we could assume that at most ϵn symbols get erased for some $\epsilon \in (0, 1)$.

2 Parity Checks and the Hamming Code

Parity check.

- A *parity check* of a sequence of bits $b_1, \ldots, b_m$ is an additional bit equaling 1 if the number of 1's in $b_1 \ldots, b_m$ is odd, and 0 if the number of 1's is even.
  - Hence, in either case, there is an even number of 1's in the sequence $b_1, \ldots, b_m c$, where c is the parity check bit

- We can express this via modulo-2 arithmetic: For $a, b \in \{0, 1\}$, let the $\oplus$ operator be defined as

$$0 \oplus 0 = 1 \oplus 1 = 0$$
$$0 \oplus 1 = 1 \oplus 0 = 1$$

  (i.e., $\oplus$ means XOR.) Then the parity check of $b_1, \ldots, b_m$ is $c = b_1 \oplus \ldots \oplus b_m$.

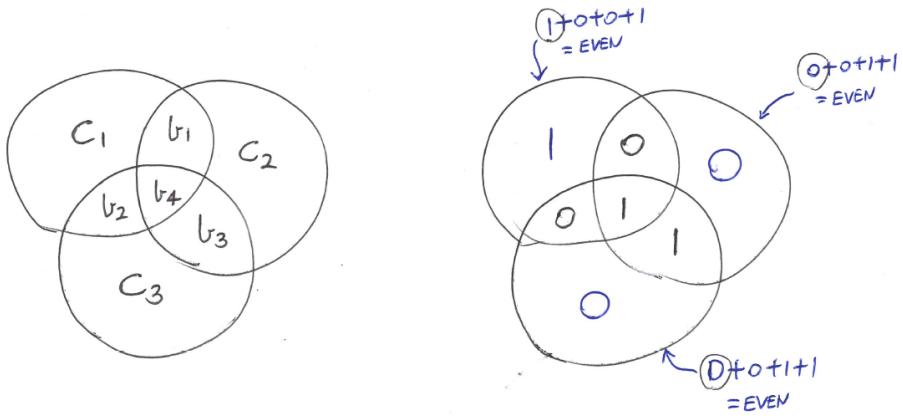
- If we transmit the length-$(m+1)$ sequence $b_1 \ldots b_m c$ across a channel, and one of the bits is flipped, we will notice that the total number of 1's is no longer even. Hence, we can *detect* a single bit flip. However, a little thought reveals that we cannot *correct* it.

- The idea that permits error correction: *Send multiple parity checks applied to different groups of bits*

Simple examples.

- Perhaps the simplest type of code is the *repetition code*, which (for example) encodes 1010 into 111000111000. By a majority vote rule, this permits the *correction* of one bit flip in each of the groups of 3 bits. We can also repeat more than 3 times for improved reliability, but this quickly becomes extremely inefficient in terms of the communication rate.

- A famous example of a less trivial code is the *Hamming code*, which (for the simplest version of it) maps 4 bits to 7 bits and can correct a single bit flip:



  Here the original bits are $b_1 b_2 b_3 b_4$, and the check bits $c_1 c_2 c_3$ are chosen to make the number of 1's per circle even. In the above terminology, each c_i is a *parity check* of the b_i's in the corresponding circle.

  - Note: When we talk about being able to correct a certain number of bit flips, this may include flips in both the uncoded bits b_i and the check bits c_i.
  - How to correct one bit flip: Observe that each possible single bit flip changes a unique combination of circles from "even number of 1s" to "odd number of 1s" (e.g., the middle bit changes all 3 circles; the top-left bit only changes the top-left circle). Therefore, the decoder can check which circles have an odd number of 1s, and un-flip the corresponding bit. If the decoder sees all 3 circles already having an even number of 1s, then no changes are made.

3 Linear Codes

Recall from the first section that k is the number of message bits, n is the number of codeword bits, and $R = \frac{k}{n}$ is the communication rate.

Definition and generator matrix.

- For reasons to be made clear shortly, we say that any code comprised of parity checks is a *linear code*.
- We distinguish between the following two cases:
  - A **systematic parity-check code** is one in which the first k (out of n) bits of $\mathbf{x}$ are always precisely the original k bits, and the remaining $n-k$ bits are parity checks:

$$x_i = u_i, \quad i = 1, \ldots, k,$$
$$x_i = \bigoplus_{j=1}^{k} u_j g_{j,i}, \quad i = k+1, \ldots, n$$

    where $g_{j,i} = 1$ if the parity check in location i includes u_j, and $g_{j,i} = 0$ otherwise. For instance, the Hamming code described above is a systematic parity-check code.
  - A **general parity-check code** is one in which all n codeword bits may be arbitrary parity checks:

$$x_i = \bigoplus_{j=1}^{k} u_j g_{j,i}, \quad i = 1, \ldots, n.$$

    Clearly a systematic code is a special case of this, since it corresponds to setting $g_{j,i} = \mathbf{1}\{j = i\}$ for $i = 1, \ldots, k$.
- The above (general) formula for generating each x_i from $u_1, \ldots, u_k$ can be succinctly summarized as a (modulo-2) vector-matrix multiplication:

$$\mathbf{x} = \mathbf{uG}, \quad \text{(in modulo-2 arithmetic)}$$

  where $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{u} = (u_1, \ldots, u_k)$ are the suitable row vectors, and

$$\mathbf{G} = \begin{bmatrix} g_{1,1} & g_{1,2} & \cdots & g_{1,n} \\ g_{2,1} & g_{2,2} & \cdots & g_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ g_{k,1} & g_{k,2} & \cdots & g_{k,n} \end{bmatrix}$$

  is known as the *generator matrix*.
  - Interpretation: The 1's in each column indicate which bits are included in the parity check

- In the special case of a systematic code, this simplifies to

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & \dots & 0 & g_{1,k+1} & \cdots & g_{1,n} \\ 0 & 1 & \dots & 0 & g_{2,k+1} & \cdots & g_{2,n} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & g_{k,k+1} & \cdots & g_{k,n} \end{bmatrix}$$

  with the left-most $k \times k$ sub-matrix being the identity matrix.

- With the mapping from $\mathbf{u}$ to $\mathbf{x}$ being described by the matrix multiplication $\mathbf{x} = \mathbf{uG}$ (in modulo-2 arithmetic), we can now justify the terminology *linear code*: If $\mathbf{u}$ and $\mathbf{u}'$ are two different message sequences, and their corresponding codewords are $\mathbf{x} = \mathbf{uG}$ and $\mathbf{x}' = \mathbf{uG}'$, then

$$\begin{aligned} \mathbf{x} \oplus \mathbf{x}' &= \mathbf{uG} \oplus \mathbf{u}'\mathbf{G} \\ &= (\mathbf{u} \oplus \mathbf{u}')\mathbf{G}, \end{aligned}$$

  which means that $\mathbf{x} \oplus \mathbf{x}'$ is also a codeword (corresponding to message $\mathbf{u} \oplus \mathbf{u}'$). In other words, the (modulo-2) sum of any two valid codewords is another valid codeword.

  - Note: We have extended the $\oplus$ notation to vectors/sequences, which is done via a bit-by-bit application of the definition above, e.g., $0001 \oplus 1101 = 1100$ and $101 \oplus 010 = 111$.

- **Examples.** With $k = 4$, the generator matrices for the single-parity-check code and Hamming code (described at the start of the lecture) are given by

$$\mathbf{G}_{\text{parity}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}, \qquad \mathbf{G}_{\text{Hamming}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}. \tag{1}$$

  Sometimes you might see the latter with the last 3 columns in a different order, but re-ordering columns just amounts to re-labeling the indices of parity check bits.

  - Pre-multiplying $\mathbf{G}_{\text{Hamming}}$ by all 16 possible $\mathbf{u}$ sequences, we can list all the codewords of the Hamming code:

$$\begin{array}{cccc} 0000000 & 0001111 & 0010011 & 0011100 \\ 0100101 & 0101010 & 0110110 & 0111001 \\ 1000110 & 1001001 & 1010101 & 1011010 \\ 1100011 & 1101100 & 1110000 & 1111111. \end{array}$$

  - For instance, the codeword 1100011 is obtained from $\mathbf{u} = 1100$ by taking the modulo-2 sum of the first two rows of $\mathbf{G}_{\text{Hamming}}$. For $\mathbf{u} = 1111$, we take the modulo-2 sum of all 4 rows.

- We briefly state without proof that any general code can be reduced to an "equivalent" (same codewords in a different order) systematic code by applying a technique similar to Gaussian elimination on $\mathbf{G}$.

Parity-check matrix.

- A matrix closely related to $\mathbf{G}$, but which will be more directly useful when it comes to decoding, is called the *parity-check matrix* $\mathbf{H}$. It is an $n \times (n-k)$ matrix that satisfies

$$\mathbf{xH} = \mathbf{0} \iff \mathbf{x} \text{ is a valid codeword.}$$

  Notice the distinction:

  - $\mathbf{G}$ is used to *generate* $\mathbf{x}$ from $\mathbf{u}$;
  - $\mathbf{H}$ is used to *check* if $\mathbf{x}$ can be generated from *any* $\mathbf{u}$ (doing this naively using $\mathbf{G}$ by testing all 2^k possible $\mathbf{u}$ would be grossly inefficient).

- While such check matrices exist for all generator matrices, we will focus our attention on the systematic case, as it is much simpler. Recall the two formulas we gave for x_i (in terms of the $\{u_j\}$ and $\{g_{j,i}\}$) in the systematic case; substituting the first into the second gives

$$x_i = \bigoplus_{j=1}^{k} x_j g_{j,i}, \quad i = k+1, \ldots, n.$$

  Adding x_i (modulo 2) to both sides, the left-hand side gives $x_i \oplus x_i = 0$, and we are left with

$$\left(\bigoplus_{j=1}^{k} x_j g_{j,i} \right) \oplus x_i = 0, \quad i = k+1, \ldots, n.$$

  Converting to matrix form reveals that the following choice of $\mathbf{H}$ indeed gives $\mathbf{xH} = \mathbf{0}$:

$$\mathbf{H} = \begin{bmatrix} g_{1,k+1} & g_{1,k+2} & \cdots & g_{1,n} \\ g_{2,k+1} & g_{2,k+2} & \cdots & g_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ g_{k,k+1} & g_{k,k+2} & \cdots & g_{k,n} \\ 1 & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 1 \end{bmatrix}.$$

  It is also not hard to establish the opposite, i.e., if $\mathbf{xH} = \mathbf{0}$ then $\mathbf{x}$ is indeed a valid codeword.

- Stated more succinctly, we have

$$\mathbf{G} = [\mathbf{I}_k \;\; \mathbf{P}] \quad \Longrightarrow \quad \mathbf{H} = \begin{bmatrix} \mathbf{P} \\ \mathbf{I}_{n-k} \end{bmatrix},$$

  where $\mathbf{I}_m$ is the $m \times m$ identity matrix, and $\mathbf{P}$ is the remaining $k \times (n-k)$ submatrix of $\mathbf{G}$.

- **Examples.** The check matrices corresponding to (1) are

$$\mathbf{H}_{\text{parity}} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \qquad \mathbf{H}_{\text{Hamming}} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

- To give a flavor of why the check matrix is useful for decoding, notice that if $\mathbf{y} = \mathbf{x} \oplus \mathbf{z}$ with $\mathbf{z}$ indicating which bits got flipped, then

$$\begin{aligned} \mathbf{yH} &= (\mathbf{x} \oplus \mathbf{z})\mathbf{H} \\ &= \big(\mathbf{xH}\big) \oplus \big(\mathbf{zH}\big) \\ &= \mathbf{zH}, \end{aligned} \tag{2}$$

  where we first used linearity, and then the fact that $\mathbf{xH} = \mathbf{0}$ for any valid codeword $\mathbf{x}$.

  - In particular, if $\mathbf{z}$ only contains a single 1 (i.e., only one bit got flipped), then $\mathbf{yH}$ is simply the i-th row of $\mathbf{H}$, where i is the index of the flipped bit.
  - But notice that in $\mathbf{H}_{\text{Hamming}}$, all the rows are distinct! This means that by looking at $\mathbf{H}_{\text{Hamming}}$, we can immediately identify which bit got flipped and therefore correct it.

Notes on storage and computation.

- Notice that the code is fully specified by $\mathbf{G}$ (or $\mathbf{H}$), which only requires nk bits of storage.
- However, efficient decoding (getting from the noisy channel output $\mathbf{y}$ back to $\mathbf{u}$) is still challenging, and the choice of $\mathbf{G}$ plays a crucial role in how efficiently can be done.
- In fact, at this stage it is unclear whether any $\mathbf{G}$ with good error correcting capabilities even exists! This is addressed in Chapter 14 of MacKay's book (optional reading), where it is shown that a *random generator matrix* $\mathbf{G}$ (i.e., each entry is 1 or 0 with equal probability) achieves arbitrarily small error probability on the BSC at all rates below capacity, when used in conjunction with an optimal decoder. Thus, **very good linear codes exist** for optimal decoders. (Achieving the same with a *computationally efficient decoder* requires more work...)

4 Distance Properties

We will only touch on the basics of distance properties here; if you are interested in knowing more, see Chapter 13 of MacKay's book.

Definition and properties.

- **Definition 1.** The *Hamming distance* between two vectors $\mathbf{x} = (x_1, \ldots, x_n)$ an $\mathbf{x}' = (x'_1, \ldots, x'_n)$ (having the same length n) is the number of positions in which they differ:

$$d_{\mathrm{H}}(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{n} \mathbb{1}\{x_i \neq x'_i\}.$$

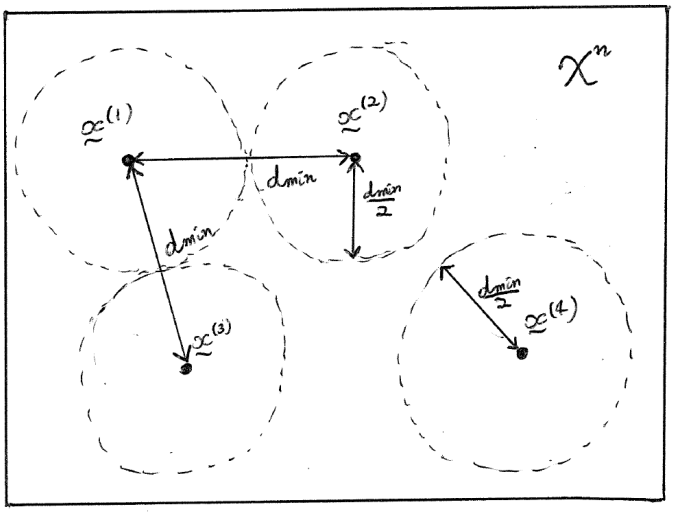
  For instance, the Hamming distance between 00110011 and 00010111 is 2.

- **Definition 2.** The *minimum distance* of a codebook $\mathcal{C}$ of length-n codewords is

$$d_{\min} = \min_{\mathbf{x} \in \mathcal{C}, \mathbf{x}' \in \mathcal{C} \,:\, \mathbf{x} \neq \mathbf{x}'} d_{\mathrm{H}}(\mathbf{x}, \mathbf{x}').$$

  Intuitively, we should expect higher $d_{\min}$ to mean better robustness to noise in the channel.

- The following illustration shows that if the minimum distance is $d_{\min}$, then (at least given enough computation time) it is possible to *correct up to* $d_{\min} - 1$ *erasures* and *correct up to* $\frac{d_{\min}-1}{2}$ *bit flips*:



  - For correcting erasures: Simply note that if $d_{\min} - 1$ code symbols are replaced by an erasure symbol '?', then there is only one way to fill them in to get a valid codeword (otherwise, the minimum distance would be $d_{\min} - 1$ or less!)
  - For correcting flips: Note that the balls of radius $\frac{d_{\min}-1}{2}$ around each codeword cannot overlap. This means that if we decode each $\mathbf{y}$ by mapping to the closest codeword, we will always be correct if at most $\frac{d_{\min}-1}{2}$ flips occurred.

- **Claim.** If $\mathcal{C}$ is the set of codewords formed by a given linear code with $d_{\min} > 0$,[3] then

$$d_{\min} = \min_{\mathbf{x} \in \mathcal{C} \,:\, \mathbf{x} \neq \mathbf{0}} w(\mathbf{x}),$$

  where $w(\mathbf{x}) = \sum_{i=1}^{n} \mathbb{1}\{x_i = 1\}$ is the *weight* of the codeword $\mathbf{x}$. Hence, for linear codes, the minimum distance equals the minimum weight.

[3] A code with $d_{\min} = 0$ is a terrible idea – it means two different vectors of information bits $\mathbf{u}, \mathbf{u}'$ lead to the same codeword!

– Proof: Let $\mathbf{x}', \mathbf{x}''$ be the two codewords at a minimum distance from each other. Then by linearity, $\mathbf{x}' \oplus \mathbf{x}''$ is also a valid codeword, and its weight is precisely $d_{\mathrm{H}}(\mathbf{x}', \mathbf{x}'') = d_{\min}$. In addition, the assumption $d_{\min} > 0$ implies that it is not the all-zero codeword.

Conversely, since $\mathbf{x} = \mathbf{0}$ is always a valid codeword (corresponding to $\mathbf{u} = \mathbf{0}$), any codeword weight also corresponds to a distance (to the all-zero codeword).

- **Example.** Recall the 16 codewords we listed earlier for the Hamming code. The minimum non-zero weight (and hence minimum distance) is 3, which is consistent with the fact that the Hamming code can correct $\frac{3-1}{2} = 1$ bit flip.

- **Notes on more general codes.**

  – Naturally, there is an inherent trade-off between achieving a high minimum distance (i.e., high $d_{\min}$) and a high rate (i.e., high $R = \frac{k}{n}$). Understanding this trade-off has been a major aspect of coding theory for decades, with two famous examples being the Gilbert-Varshamov bound (achievability / existence) and the Sphere Packing bound (converse / impossibility), which you can look up if you are interested.

  – When designing practical codes, even the following goal is non-trivial: *As $n \to \infty$, achieve a positive rate (i.e., $\frac{k}{n} \to R^* > 0$) and a constant-fraction minimum distance (i.e., $\frac{d_{\min}}{n} \to \delta^* > 0$).*

  – In Section 6 we will cover a particularly well-known class known as Reed-Solomon Codes with very strong distance properties, and in the next lecture we will show that a class called Expander Codes can also achieve the above goal.

(Optional**) Distance vs. capacity.**

- While a high minimum distance seems like a nice property to have, it corresponds to a fundamentally different modeling assumption compared to the channel capacity:

  – The minimum distance goal is aligned with *worst-case errors*, where (for instance) we need to be able to correct *arbitrary patterns* of up to δn bit flips introduced by the channel.

  – The channel capacity goal is aligned with *random errors*, where (for instance) we need to be able to correct bit flips that occur *independently* with probability δ each.

  Neither of these goals should be viewed as "better" than the other in general; either may be preferable depending on the application.

- Of course, it is worth asking whether achieving capacity and attaining good distance properties are actually equivalent goals. However, as argued in Chapter 13 of MacKay's book, this is not the case:

  – The best possible rate with minimum distance δn is at most the channel capacity with *double* the noise level, 2δ;

  – The channel capacity is positive for any $\delta \in \left(0, \frac{1}{2}\right)$, but no positive rate can be achieved for $\delta > \frac{1}{4}$ if one insists on a minimum distance δn;

  – Examples are known where the channel capacity is achieved despite a "bad" minimum distance.

- Although distinct from achieving capacity, the design of codes with good distance properties is a very important problem in its own right. Examples of such codes include BCH codes (a generalization of Hamming codes), and Reed-Solomon codes, the latter of which we cover below.

5 Decoding Techniques

Decoding is the aspect of linear codes that is much harder to make efficient (while maintaining strong guarantees and/or practical performance). We will not explore this aspect in significant detail (see Section 7 for some pointers to topics that cover this).

In this section, we briefly outline some decoding rules that are *optimal* (in some sense to be described) but *computationally inefficient* (unless the number of bit flips is very small).

Maximum-likelihood decoding:

- Suppose that the output $\mathbf{y}$ is produced from $\mathbf{x}$ via a probabilistic model $P_{\mathbf{Y}|\mathbf{X}}$.
- Then, it can be shown (proof omitted) that if the message bits are uniformly random, the following decoding rule is optimal in the sense of minimizing the error probability $P_{\mathrm{e}} \mathbb{P}[\hat{\mathbf{u}} \neq \mathbf{u}]$:

$$\hat{\mathbf{u}} = \arg\max_{\mathbf{u}'} P_{\mathbf{Y}|\mathbf{X}}(\mathbf{y}|\mathbf{x}(\mathbf{u}')),$$

  where $\mathbf{x}(\mathbf{u}')$ is the codeword corresponding to $\mathbf{u}'$. This is true even for non-linear codes.
- Unfortunately, the above maximum is over 2^k choices, so checking all of these is prohibitive unless k is very small.

Minimum distance decoding:

- For the binary symmetric channel (BSC), we can write $P_{\mathbf{Y}|\mathbf{X}}(\mathbf{y}|\mathbf{x})$ as a product $\prod_{i=1}^{n} P_{Y|X}(y_i|x_i)$, where each $P_{Y|X}(y|x_i)$ is equal to $1-\delta$ if $y = x_i$, and δ otherwise.
- Using this, it can fairly easily be shown that the maximum-likelihood decoding rule becomes equivalent to the following for the BSC when $\delta < \frac{1}{2}$:

$$\hat{\mathbf{u}} = \arg\min_{\mathbf{u}'} d_{\mathrm{H}}(\mathbf{x}(\mathbf{u}'), \mathbf{y}).$$

  This is known as *minimum distance decoding*.
- Minimum distance decoding is also highly desirable under the non-probabilistic (distance-based) viewpoint – if there are at most $\frac{d_{\min}-1}{2}$ bit flips, then minimum distance decoding is guaranteed to recover the correct codeword.
- However, like with maximum-likelihood decoding, the computational cost may be highly prohibitive.

Syndrome decoding.

- Syndrome decoding is a technique that takes the idea of minimum distance one step further in the special case of linear codes.
- Recall the evaluation of $\mathbf{yH}$ given above:

$$\begin{aligned} \mathbf{yH} &= (\mathbf{x} \oplus \mathbf{z})\mathbf{H} \\ &= (\mathbf{xH}) \oplus (\mathbf{zH}) \\ &= \mathbf{zH}, \end{aligned} \tag{3}$$

with the middle step using the fact that $\mathbf{xH} = \mathbf{0}$ for any valid codeword $\mathbf{x}$.

- The quantity
$$\mathbf{S} = \mathbf{zH} = \mathbf{yH}$$
is called the *syndrome*, and we have just shown that it can immediately be computed from the check matrix $\mathbf{H}$ given the channel output $\mathbf{y}$.

- It can fairly easily be shown that for a linear code, if the syndrome is $\mathbf{S}$, then the minimum-distance codeword to $\mathbf{y}$ can be obtained by finding
$$\hat{\mathbf{z}} = \underset{\tilde{\mathbf{z}} \,:\, \tilde{\mathbf{z}}\mathbf{H}=\mathbf{S}}{\arg\min}\, w(\tilde{\mathbf{z}})$$
(where $w(\tilde{\mathbf{z}})$ is the number of 1's in $\tilde{\mathbf{z}}$) and then computing
$$\hat{\mathbf{x}} = \mathbf{y} \oplus \hat{\mathbf{z}}.$$
In addition, if the code is systematic, the estimate $\hat{\mathbf{u}}$ of the original information bits can then be formed by taking the first k entries of $\hat{\mathbf{x}}$.

- Using syndrome decoding, if very few bit flips occurred during transmission, then we may avoid searching over an exponentially large number of codewords – there are only n vectors $\tilde{\mathbf{z}} \in \{0,1\}^n$ of weight 1, $\binom{n}{2}$ of weight 2, and so on. Since we are minimizing $w(\cdot)$, we can start from the smallest weights, work upwards, and stop once a match is found.

- On the other hand, if a significant number of bit flips occur during transmission, this decoding method still typically requires far too much computation to be practical. But the general idea still forms the basis of certain very powerful computationally efficient codes.

Interested students may refer to `https://www.comp.nus.edu.sg/~scarlett/CS3236_notes/06-Codes.pdf` for the proofs that were omitted in this section.

6 Reed-Solomon Codes

This section gives an overview Reed-Solomon Codes, which are of one of the most widely-used distance-based codes (i.e., codes designed to have high $d_{\min}$). These are used extensively throughout theoretical computer science, as well as practical applications such as DVD and QR codes.

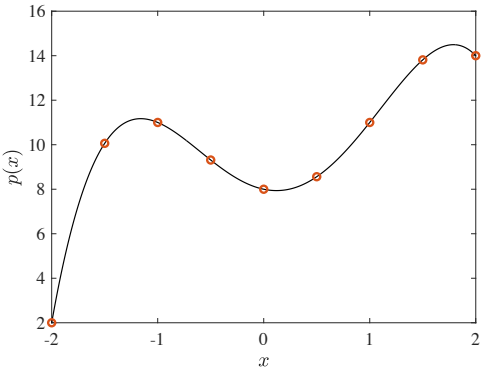
Unlike what we covered in the previous sections, these codes are *non-binary*, i.e., the symbols of the length-n codewords take values in an alphabet with (significantly) more than two symbols. We will later discuss methods for converting a non-binary code to a binary code.

6.1 Intuition

Here we give the rough idea behind Reed-Solomon Codes using concepts that should be more familiar (but can't be used directly).

Imagine that we want to communicate 5 *real-valued* numbers u_0, u_1, u_2, u_3, u_4 from a sender to a receiver, but whenever a number is sent, there is a chance that it gets "erased" and the receiver instead sees a special symbol '?' to indicate that. The following strategy gives a way that we could combat such erasures:

- Interpret u_0, u_1, u_2, u_3, u_4 as *coefficients of a degree-4 polynomial*: $p(x) = u_0 + u_1 x + u_2 x^2 + u_3 x^3 + u_4 x^4$.
- Pick *more than 5* points x to evaluate the polynomial at, say $x \in \{-2, -1.5, \ldots, 0, \ldots, 1.5, 2\}$, e.g.:



- The encoder sends the corresponding $p(\cdot)$ values, e.g., $p(-2), p(-1.5), \ldots, p(2)$ corresponding to the red circles in the above figure.
- The receiver receives a length-9 output, but some of the values are replaced by '?' to mean 'erasure'.
- However, as long as 5 (or more) of them are non-erased, the decoder can recover $p(x)$ – this is because *a degree-d polynomial is uniquely specified by any $d+1$ points*. Since there is a one-to-one correspondence between $p(\cdot)$ and $(u_0, u_1, u_2, u_3, u_4)$, recovering the former means recovering the latter.

Naturally, if some of the $p(\cdot)$ values get *altered* rather than *erased*, then recovery might be more difficult, but mathematically it could still be possible, especially if we increase the number '9' of evaluations above. For example, one could design the decoder based on recovering many polynomials, each corresponding to a different subset of 5 received values. If the number of altered values is small enough, then the polynomial that is recovered most often should be the correct one.

What is perhaps most unrealistic in the above description is the notion of sending and receiving (and performing computation on) infinite-precision real numbers. The idea of Reed-Solomon codes is to use the same idea, but with *finite fields instead of the real number field*. The use of a finite field makes it plausible to implement on a computer.

6.2 Finite Fields

- Mathematically, a field is a set of "numbers" together with addition ($+$) and multiplication ($\times$) operations that satisfy natural axioms that we mostly won't delve into (e.g., $a \times b = b \times a$). Importantly, there should be a "zero element" (usually denoted by 0) such that $a + 0 = a$, and an "identity element" (usually denoted by 1) such that $a \times 1 = a$. Moreover, every non-zero element a should have some "multiplicative inverse" a^{-1} such that $a \times a^{-1} = 1$, and some "additive inverse" $(-a)$ such that $a + (-a) = 0$.
- The set of real numbers with the usual $+$ and $\times$ forms a field. The same goes for the rational numbers. The integers, however, do not form a field, because we can't have $a \times a^{-1} = 1$ for integer-valued a^{-1}.
- In previous sections, we worked with mod-2 arithmetic, in particular using $\oplus$ for addition. Multiplication is trivial in this case: $0 \times a = 0$ and $1 \times a = a$. This is known as the *two-element Galois field* or GF(2). (In this context you can take "Galois" to be synonymous with "finite".)

- More generally, the numbers $\{0, 1, \ldots, p-1\}$ with mod-p arithmetic form a field whenever p is a prime number. For example, if $p = 7$, then we can pick any element (e.g., 3) and find an additive inverse (e.g., 4, since $3 + 4 \equiv 0 \pmod 7$) and a multiplicative inverse (e.g., 5, since $3 \times 5 \equiv 1 \pmod 7$).
- It turns out that the only possible sizes of finite fields are of the form p^m, where p is a prime number and m is a positive integer. Moreover, once such a size is specified, the field is unique (up to relabeling).
- We will proceed using only the above (limited) knowledge, but interested students can refer to Forney's "Introduction to finite fields" lecture notes (Stanford University) for a proper introduction.

6.3 Code Construction

The parameters of a Reed-Solomon code are as follows:

- The message length k and codeword length n;
- The size q of a finite field $\mathbb{F}$; as noted above, we must have $q = p^m$ for some prime p and positive integer m. We further impose the requirement $q \geq n$.

Note that the message of length k is now a sequence of *finite field elements* rather than a sequence of bits, so the total number of messages is q^k rather than 2^k.

Then, similar to what we did with real numbers in Section 6.1, the code construction is as follows:

- Fix a set of n distinct *evaluation points* $a_1, a_2, \ldots, a_n$ in $\mathbb{F}$. Since we assumed $q \geq n$, we can do this in a way such that the a_i are all distinct. There is flexibility in the specific choice of a_i's; a common choice is to let them be of the form $1, \alpha, \alpha^2, \ldots, \alpha^{n-1}$ for suitably-chosen $\alpha \in \mathbb{F}$.
- Given the message symbols $u_0, \ldots, u_{k-1}$ (each in $\mathbb{F}$), define the following polynomial:

$$p(x) = \sum_{i=0}^{k-1} u_i x^i,$$

  where here and subsequently, all arithmetic is done over the finite field $\mathbb{F}$.
- The length-n codeword is then simply $(p(a_1), p(a_2), \ldots, p(a_n))$, i.e., we evaluate the polynomial at the n evaluation points.

This defines a *linear code*, where now the notion of *linear* is a bit more general than the binary case: If $\mathbf{c}$ and $\mathbf{c}'$ are both codewords, then so is $\alpha\mathbf{c} + \beta\mathbf{c}'$ for any $\alpha \in \mathbb{F}$ and $\beta \in \mathbb{F}$. *(Exercise: Try to prove this.)*

Much like in the real-valued case in Section 6.1, dealing with *erasures* is easiest: Since we used a degree-$(k-1)$ polynomial, we can recover the polynomial (and hence the message sequence) as long as k or more symbols remain non-erased. On the other hand, if some of the code symbols are *substituted* by other elements of $\mathbb{F}$ (rather than 'erased'), decoding becomes more difficult.

Having said that, various computationally efficient algorithms are known for decoding Reed-Solomon codes with substitution noise, such as Berlekamp-Welch with time $O(n^3)$, Berlekamp-Massey which is connected to syndrome decoding, and other approaches based on the Fast Fourier Transform. The details of how these decoders work is beyond the scope of this course.

6.4 Properties

In this section, we state and prove some of the most important properties of Reed-Solomon codes. Analogous to the binary case, we define $d_{\min}$ to be the minimum distance (i.e., number of differing symbols) between two codewords.

Claim. The minimum distance of a Reed-Solomon code is $d_{\min} = n - k + 1$.

Proof:

- Like in the binary case, we start with the fact (with a similar proof) that the minimum distance equals the minimum non-zero weight, where "weight" now means "number of non-zero symbols".
- The minimum non-zero weight is equal to $n - \tau$, where τ equals the maximum number of zeros in a non-zero codeword.
- Then, the following fact about polynomials in $\mathbb{R}$ carries over to polynomials on finite fields: *Any non-zero polynomial of degree d has at most d zeros.*
- Since $p(x)$ has degree $k - 1$, this implies $\tau \leq k - 1$ and hence $d_{\min} \geq n - k + 1$. The next claim below is sufficient for deducing that this is not only a lower bound but in fact an equality.

Claim. For any (n, k, q) with $q \geq n$, the minimum distance of a Reed-Solomon code is optimal; no code from $\mathbb{F}^k$ to $\mathbb{F}^n$ can attain a minimum distance higher than $n - k + 1$. *(Note: This is known as the Singleton Bound, or at least a certain form of it)*

Proof:

- Fix an arbitrary code, and suppose that its minimum distance is $d_{\min} = d$.
- If we delete the first $d - 1$ symbols from each codeword, the list of remaining shortened codewords must still all be distinct (due to the minimum distance being d).
- But the number of such codewords is at most q^{n-d+1}, since each symbol takes on one of q values.
- For for a code from $\mathbb{F}^k$ to $\mathbb{F}^n$, the number of codewords is exactly q^k. Comparing to the previous dot point, we get $q^k \leq q^{n-d+1}$, hence $k \leq n - d + 1$, hence $d \leq n - k + 1$.

Discussion.

- So if Reed-Solomon codes have optimal distance properties and can be implemented in a computationally efficient manner, why don't we use them everywhere?
- The answer is that they have a significant disadvantage: *Requiring a large alphabet size ($q \geq n$).*
- The reason this is a disadvantage is that real-world codes typically need to be binary, or need to be some other size (typically much smaller than n) according to the communication channel being used.
- We will next discuss how to "convert" a Reed-Solomon code into a binary code. Unfortunately doing so comes at the cost of losing the property of having the best possible minimum distance, but it can still lead to a very good code.

6.5 From Non-Binary to Binary

A codeword in a Reed-Solomon is a sequence of non-binary characters; for concreteness, let's say that they come from a size-25 set of symbols that we label as $\{A, B, C, \ldots, X, Y\}$. But if we want to transmit over a binary channel, or store the codeword on a DVD or QR code (etc.), we need to use binary symbols. How can we use Reed-Solomon codes in such scenarios?

The fact that the message symbols $(u_1, \ldots, u_k)$ above are non-binary is not a significant issue; what's more important is making the *codeword symbols* binary. To do so, the basic idea is to *map each non-binary codeword symbol to a binary sequence*, e.g.:

- We could map $A \to 10\ldots0$, $B \to 010\ldots0$, and so on, up to $Y \to 0\ldots01$. This is called *one-hot encoding*. This strategy can be sufficient in certain cases, but it is typically far from ideal, since the code length is blown up by the alphabet size q, leading to a low communication rate.
- If the alphabet size is q, we could map each letter to a unique length-$\lceil \log_2 q \rceil$ binary sequence. This means that the code length only grows according to the *logarithm* of the alphabet size. However, this may not be ideal in terms of error resilience: Within each length-$\lceil \log_2 q \rceil$ block, *any* one of those bits getting flipped amounts to having a symbol substitution in the non-binary code.
- To better balance the goals of error-resilience and not increasing the code length too much, the best approach is usually to *use another (small) error-correcting code to map each non-binary symbol to a binary sequence*. For example, one might use a Hamming-like code that can at least correct a small number of bit flips. This is known as *concatenated coding*, with the Reed-Solomon code being the "outer code" and the binary code being the "inner code".

To name just one specific example, the Justesen code (`https://en.wikipedia.org/wiki/Justesen_code`) is a binary code based on Reed-Solomon coding and concatenation, and it comes with a simple and explicit trade-off between the rate and the minimum distance.

To re-iterate, the conversion of Reed-Solomon codes to binary codes invariably loses the "optimal minimum distance" property stated above, but it often gives something that is still very good.

7 (**Optional**) Other Coding Methods

7.1 Distance-Based Codes

Here we mention just one other famous example beyond Reed-Solomon codes: *Bose-Chaudhuri-Hocquenghem* (BCH) codes are also based on polynomials on finite fields, and they can also be viewed as a generalization of Hamming codes.

Very briefly, the idea is to interpret the message as a polynomial (e.g., in the binary case, interpret 010011 as $x^4 + x + 1$ if the bits are listed from most significant to least significant) and multiply it by some *generator polynomial* $g(x)$ (using finite field arithmetic) to produce another polynomial – its coefficients give the codeword. A suitably-chosen $g(x)$ can ensure good distance separation, and there are also variants that ensure the resulting code is systematic.

BCH codes are well-defined over general finite field sizes (in particular, we have the option of sticking to binary), allow precise control over the minimum distance, and are generally considered easier to decode (using syndrome decoding methods) than Reed-Solomon codes. They have found widespread use in applications such as satellite communication, storage devices, and QR codes.

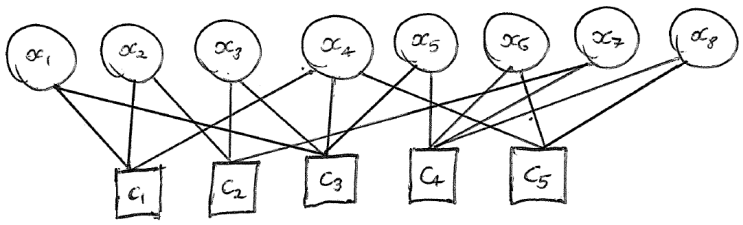
7.2 Capacity-Achieving Codes

History.

- The channel coding theorem was published in 1948. This spawned decades of research on practical coding techniques, but none were seen to come close to achieving capacity for a long time.
- In the early 1990s, *turbo codes* were (empirically) seen to be very close to achieving capacity.
- *Low density parity check* (LDPC) codes were invented in the 1960s, but perhaps due to limited computing power, they remained unused for a long time. After the invention of turbo codes, they were rediscovered and also shown to be nearly capacity-achieving.
- *Polar codes* are a more recent development (2008). They are much nicer to analyze theoretically, in particular to rigorously establish that they are capacity-achieving. Initially they weren't as good in practice, but now they have caught up with (and in same cases even "overtaken") the others and are used extensively.

(Very) Brief overview of LDPC codes.

- As a rough definition, an LDPC code is a (typically non-systematic) linear code such that the check matrix $\mathbf{H}$ contains mostly 0's and relatively few 1's.
- We can picture this via a *factor graph* representation:



  Here circles correspond to the n codeword bits ("variable nodes"), and squares correspond to the $n-k$ parity equations ("check nodes"), i.e., if there are no errors, there should be an even number of 1's connected to each square. The *low density* assumption amounts to saying that the above graph is *sparse* (has few edges).
- The reason sparsity / low density is useful is that it permits effective *belief propagation* decoding, in which information is shared between variable nodes and check nodes until accurate beliefs (posterior probability estimates) on the transmitted bits are obtained.
- Strictly speaking, this is only guaranteed to end up with accurate posterior estimates when the underlying graph has no cycles (i.e., one can never follow any path of edges and end up back at the starting node). Although sparse graphs like the ones above do have cycles, they have *very few short cycles*, and for practical purposes this is often nearly as good as having no cycles.
- For a much better introduction to LDPC codes, see Chapter 47 of MacKay's book (see also `https://www.youtube.com/watch?v=RWUxtGh-guY` for a brief introduction in Layman's terms).

(Very) Brief overview of polar codes.

- The rough idea of polar codes is as follows:
  - First consider 2 uses of the channel, $(X_1, X_2) \to (Y_1, Y_2)$.
  - The idea: Apply some pre-processing to (X_1, X_2) to create a related channel $(\widetilde{X}_1, \widetilde{X}_2) \to (Y_1, Y_2)$ such that (i) $I(X_1, X_2; Y_1, Y_2) = I(\widetilde{X}_1, \widetilde{X}_2; Y_1, Y_2)$ (no information is lost); (ii) Comparing the new channel to the original channel, one "channel use" is less noisy, and the other is more noisy.
  - Repeat this procedure several times, moving from 2 to 4 uses of the channel, then 8 uses of the channel, then 16, etc., with many "channel uses" getting less and less noisy, and the rest getting more and more noisy.
  - *For a symmetric binary channel with capacity $C \in (0, 1)$, repeating this procedure eventually leads to a fraction C of "near-perfect" channel uses, and a fraction $1 - C$ of "near-useless" channel uses.*
  - Perfect and useless channels are very easy to handle! If it were truly perfect, we could just send a bit and receive it without noise. If it were truly useless, we would simply avoid using it at all. Polar codes do something similar, sending information only over the "near-perfect" uses.
- Extensions to non-binary and non-symmetric channels were established later.
- For a much better introduction to polar codes, see the following lecture by Emre Telatar: `https://www.youtube.com/watch?v=VhyoZSB9g0w`

CS5275 Lecture 10: Expander Graphs

Jonathan Scarlett

April 10, 2025

Acknowledgment. The first version of these notes was prepared by Niu Xinyuan and Chen Zhi Liang for a CS6235 assignment.

References:

- Video lecture: Expander Graphs by Ryan O'Donnell [1]
- Lecture notes: 7, 8, 12 by Ryan O'Donnell [2]
- Lecture notes: 3 by Ola Svensson [3]
- Textbook: Expander graphs and their applications, by Hoory, Linial, and Wigderson

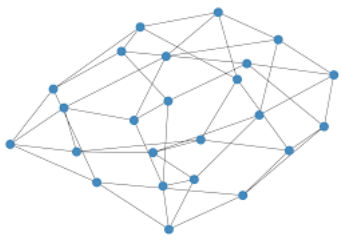
Categorization of material:

- Core material: Material before Section 4.1, except when marked 'Optional'
- Extra material: Section 5 (Applications)

(Exam will strongly focus on "Core". Take-home assessments may occasionally require consulting "Extra".)

1 Introduction

Informally, expander graphs are graphs that are simultaneously sparse and highly connected. This seems like a contradiction at first glance, but we will show formally that there are graphs that fulfill both properties. The following figure gives a simple example – there are relatively few edges, but there are still several short paths from any given node to any other.



[1] https://www.youtube.com/watch?v=bONlIjZRJhA
[2] https://www.cs.cmu.edu/~odonnell/toolkit13/
[3] https://theory.epfl.ch/courses/topicstcs/Lecture3.pdf

A direct practical motivation for this would be if we were designing a network (e.g., of inter-connected computing devices) where forming connections (edges) is expensive but we still want a high degree of connectivity. As less obvious applications, we will later see that expander graphs are also useful for reducing the number of random bits needed in derandomization tasks, and for designing error-correcting codes. There are also other more abstract uses in areas like complexity theory (e.g., circuit lower bounds).

One approach to proving the existence of expander graphs is to use the probabilistic method, i.e., showing that a certain random graph gives the desired properties with high probability. On the other hand, it is often preferable to be able to construct such graphs *explicitly* without randomization. We will study both kinds of constructions.

The outline for the lecture is as follows:

1. Properties and definitions of expanders.
2. Extension to bipartite expanders.
3. Existence of expanders via the probabilistic method.
4. Explicit constructions of expanders.
5. Applications: Error correcting codes and derandomization

2 Definitions of Expander Graphs

We consider a undirected, d-regular graph $G = (V, E)$ with $|V| = n$ vertices and $|E| = \frac{nd}{2}$ edges. A d-regular graph refers to graph where each degree of a vertex is d. In general, expander graphs are constructed by algorithm/network designers, and hence we have the flexibility to only look at d-regular graphs (and d-regularity typically suffices for applications). Further reading regarding irregular graphs of degree at most d can be found under Definition 9.2 in Hart et al. [2013].

2.1 Sparsity

The notion of a graph being "sparse" is relatively straightforward – its number of edges is small. Formally, we consider d-regular graphs such that d is a *constant that does not dependent* n. Thus, the number of edges is Thus, as $n \to \infty$, the number of edges in the graph increases as $O(n)$, which is much smaller than the maximum possible of $O(n^2)$ edges.

While slowly-growing scaling of d may also be of interest, e.g., $d = O(\log n)$, the constant-d regime is the most widespread and broadly applicable, corresponding to being as sparse as reasonably possible. (If we were to have $o(n)$ edges then some vertices would have to have no connections, so the graph certainly wouldn't be "well-connected".)

2.2 Highly connected

The connectivity of a graph can be quantified in several ways. We will look at 3 types of expansions, namely edge, vertex and spectral expansion, which help to formally define the notion of connectivity. We will also briefly discuss (without much detail) how these notions are connected to each other.

2.2.1 Edge expansion

Our first notion of connectivity can concerns edge properties, roughly stating that any given subset of vertices will have "not too few" edges from inside that subset to outside it. This prevents scenarios such as having a subset that is isolated from the rest of the graph, which would certainly not be "well-connected".

Definition 1. *For a graph $G = (V, E)$ with n vertices, let*

$$\phi[S] = \frac{\textit{no. of edges (u,v) with } u \in S, v \notin S}{|S|}. \tag{1}$$

Then, Cheeger's constant *is defined as*

$$\phi_G = \min_{0<|S|\leq \frac{n}{2}} \phi[S]. \tag{2}$$

The quantity $\phi[S]$ measures how well connected S is to its complement $S^c = V \setminus S$. For a single choice of S this might not tell us much about the connectivity of the entire graph, but we get that by performing a minimization over S to obtain ϕ_G. Notice that we have limited the size of $|S| \leq \frac{n}{2}$, but for sets bigger than that we can still draw conclusions about the connectivity by considering the complement set (which will have size below $\frac{n}{2}$).

To understand how ϕ_G related to connectivity, suppose that ϕ_G is lower bounded by a constant as $n \to \infty$, say $\phi_G > 0.05$. This means that for any subset S of size at most $\frac{n}{2}$, there are at least $0.05|S|$ edges from S to S^c. This is a constant fraction of the *highest feasible number*, which is $d|S|$ due to d-regularity. Thus, this condition prevents any scenario where some subset is "isolated" or has very low connectivity from the rest of the graph. (The constant 0.05 here is somewhat arbitrary and may seem low; sometimes any constant factor suffices for theoretical purposes, but sometimes we do care about getting a higher constant.)

We briefly note that a probabilistic interpretation is also possible – if we pick a random vertex in S and then traverse a random edge from that vertex, then probability of ending up outside S is at least some (small) constant. With this interpretation, ϕ_G can also be interpreted as measuring to what extent there are "bottlenecks" that prevent random traversals from exiting certain parts of the graph (higher ϕ_G means there are fewer and/or milder bottlenecks).

It is straightforward to show that $\phi_G > 0$ if and only if the graph is connected, i.e., there exists a path between any two vertices. (Consider trying this as an exercise.)

2.2.2 Vertex expansion

Along similar lines as edge expansion, one can consider counting the number of nodes in $V \setminus S$ that S connects to, instead of the number of edges between them. Thus, we are interested in the following ratio for a given set $S \subset V$:

$$\phi'[S] = \frac{|\partial(S)|}{|S|}, \tag{3}$$

where $\delta(S)$ is the set of vertices in $S^c = V \setminus S$ that are connected to one or more vertices in S (sometimes called the *outer boundary*).

Definition 2. *The* vertex expansion number *of $G = (V, E)$ is defined as*

$$\phi'_G = \min_{0<|S|\leq \frac{n}{2}} \phi'[S].$$

Observe that by this definition, we have for $|S| \leq n/2$ that

$$|\partial(S)| \geq \phi'_G |S|. \tag{4}$$

Often the definition of vertex expansion is given in this form directly with some parameter α (or ϵ), e.g.:

$$|\partial(S)| \geq \alpha|S|, \quad \text{or equivalently} \quad |\overline{N}(S)| \geq (1+\alpha)|S|, \tag{5}$$

where $\overline{N}(S) = S \cup \partial(S)$ is the boundary along with S itself.

We will focus on d-regular graphs with constant d (not growing with n), and for such graphs it is fairly simple to show that ϕ_G and ϕ'_G differ by at most a factor of d (see also the tutorial), so the two at least coincide to within a constant factor. On the other hand, constant factors can be important, and for vertex expansion there tend to be a wider variety of expansion notions that may be of interest depending on the application. To get some idea of why this might be the case, note the following for d-regular graphs:

- If S is a small set, then $\partial(S)$ could be anything from 0 to $d|S|$, leading to a ratio in $[0, d]$.
- However, if $|S| = n/2$, then $\partial(S)$ is also at most $n/2$, leading to a ratio in $[0, 1]$.

If we care about having more precise constants, then the definition of ϕ'_G above is limited in that it needs to capture both of the above regimes in a single number.

With the above in mind, we proceed to outline some variations (some of which are used in the tutorial/homework, and also in this lecture below for bipartite graphs):

- Instead of considering all $|S| \leq n/2$, we might only require $|\partial(S)| \geq \alpha|S|$ for smaller sets S, say $|S| \leq \gamma n$ for some $\gamma > 0$. In particular, having $\gamma = O(1/d)$ may better capture the first dot point above.
- Instead of using $\partial(S)$, the expansion may be with respect to $N(S)$, defined as the set of all neighbors *including those in* S, or with respect to $\overline{N}(S) = \partial(S) \cup S = N(S) \cup S$ as noted above.
- In the case d-regular graphs, instead of $|\partial(S)| \geq \alpha|S|$ or $|\overline{N}(S)| \geq (1+\alpha)|S|$, we may re-parametrize $\alpha = \epsilon D$ to get the condition $|\partial(S)| \geq \epsilon d|S|$ or $|\overline{N}(S)| \geq (1+\epsilon d)|S|$. This can be convenient because it naturally means that $\epsilon \in (0, 1)$, as opposed to $\alpha \in (0, d)$.

To elaborate on the last dot point, observe that in a d-regular graph, any set S is trivially connected to at most $d|S|$ other nodes, but the actual number may be fewer due to "collisions" (i.e., multiple nodes in S connecting to some other node j). The expansion property says that we always connect to at least $\epsilon d|S|$ for some constant $\epsilon \in (0, 1)$, meaning we are "never too far" from that maximum and thus there are "not too many collisions".

At least for theoretical purposes, having ϕ'_G (or α, ϵ, etc.) be any constant that doesn't decrease with n (e.g., 0.01) is often considered "large enough".

2.2.3 (**Optional**) Spectral expansion

Lastly, it is also possible to define well connectedness via the spectrum of the graph. This is sometimes of interest in its own right, and sometimes used a a stepping stone towards getting results regarding edge and vertex expansion.

Consider a d-regular graph with adjacency matrix $A \in \{0, 1\}^{n \times n}$, i.e., $A_{ij} = 1$ whenever i and j are connected. Instead of working directly with A, spectral properties of graphs are usually characterized by a

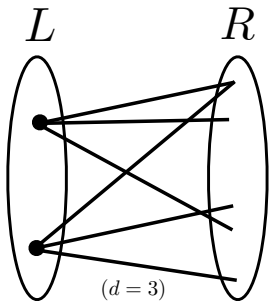


Figure 1: Bipartite graph illustration (left d-regular with $d = 3$).

related matrix called the *(normalized) Laplacian*, $L = I - \frac{1}{d}A$ (for reasons we won't go into). The matrix L has n eigenvalues, namely $0 = \lambda_0 \leq \lambda_1 \leq \cdots \leq \lambda_{n-1} \leq 2$.

In turns out that Cheeger's constant ϕ_G (introduced above for edge expansion) can be bounded in terms of λ_1 via a famous result called *Cheeger's inequality*. One reason this is useful is that computation of Cheeger's constant ϕ_G is NP-hard in general, as it involves solving for the sparsest-cut of the graph. However, the computation of the eigenvalues λ_i of L can be done efficiently.

Formally, ϕ_G is related to the second smallest eigenvalue of L, λ_1, as follows:

$$\frac{1}{2}\lambda_1 \leq \phi_G \leq 2\sqrt{\lambda_1}. \tag{6}$$

In particular, we have a direct relation to edge expansion: If λ_1 is "large", then so is ϕ_G.

Definition 3. *A (n, d, ϵ)-spectral expander graph is a d-regular graph with n vertices and $\lambda_1 \geq \epsilon$.*

We now have three definitions of expanders, which have clear differences but are all closely related, not only conceptually but also via formal connections such as Cheeger's inequality.

- **(**Optional**)** For further reading on how the expansion measures relate to each other, see, e.g., `https://people.seas.harvard.edu/~salil/pseudorandomness/expanders.pdf`

2.3 Bipartite expanders

For many of the examples in this lecture, we will talk about expanders in the context of bipartite graphs, which are graphs $G = (V, E)$ where V is partitioned into two disjoint sets L and R such that all edges are between L and R (i.e., there are no edges within L nor within R). This property is useful in applications where we are interested in mapping a set of objects to another explicitly, and hence the idea of two distinct subsets of vertices becomes relevant.

In this context, we consider *left d-regularity*, which is the property that each vertex in L is has exactly d edges to vertices in R. See Figure 1 for an illustration.

It is then appropriate to define *bipartite expanders*. In particular, we consider expansion from vertices on the *left* of the bipartite graph (instead of on every vertex, as was the case for general expanders above).

Definition 4. *A bipartite graph with two disjoint vertex sets L and R, where $|L| = n$ and $|R| = m$ and $\deg(u) = d$ for all $u \in L$, is called a $(n, m, d, \gamma, \epsilon)$ expander if for all $S \subseteq L$ with $0 < |S| \leq \gamma n$, we have:*

$$|N(S)| \geq \epsilon d|S|, \tag{7}$$

where $N(S)$ is the set of nodes in R that are connected to at least one node in S (i.e., the neighbors of S).

Note that here, for convenience, the definition is given directly in terms of the expansion condition (similar to (5)) instead of the ratio $\frac{|\partial(S)|}{|S|}$, and we use $N(S)$ which is equivalent to $\partial(S)$ for bipartite graphs. In addition, we use ϵd instead of α, since this will turn out to be more convenient in the applications of bipartiate graphs (Section 5).

The parameter $\gamma \in [0,1]$ gives some possible relaxation where we don't necessarily need the expansion property to hold for all subsets of size $\leq n$ or even $\leq n/2$, but rather only $\leq \gamma n$. This turns out to be sufficient in many applications even when γ is fairly small.

3 Existence via the Probabilistic Method

At this stage, it may not be clear whether the above expansion notions are even attainable. It turns out that they indeed are, and we can show it using the probabilistic method – generate a random graph according to a suitably-defined distribution, and show that it has a positive probability of being an expander graph.

There are many results showing the existence of expanders of various types (edge/vertex/spectral, bipartite vs. non-bipartite) and with various parameters (e.g., ϵ and/or γ). Rather than giving a general statement, for concreteness we focus here on one particular set of parameters for bipartite vertex expanders.

Theorem 3.1. *Consider $G = (L, R, E)$ with two disjoint vertex subsets L and R, where $|L| = n$ and $|R| = m$ and $\deg(u) = d$ for all $u \in L$. Suppose that G is randomly constructed as follows:*

- *For each vertex $u \in L$, perform the following independently: Select d vertices in R uniformly at random without replacement, and create an edge from u to each of these vertices.*

Then, for $d \geq 32$, $m \geq 3n/4$, and large enough n, it holds with probability at least $\frac{18}{19}$ that the graph has the following vertex expansion property:

$$|N(S)| \geq \frac{5d}{8}|S|, \quad \forall S \subseteq L : |S| \leq \frac{n}{10d}. \tag{8}$$

That is, the graph is an $\left(n, m, d, \frac{5}{8}, \frac{1}{10d}\right)$ bipartite expander.

Proof. Consider the random graph described in the theorem, and let $S \subset L$ have cardinality $s = |S| \leq \frac{n}{10d}$. Given such an S, the desired expansion probability can only be violated if $N(S) \subseteq T$ for some $T \subseteq R$ of size $t = \frac{5d}{8}s$. We call this a "bad event" and denote it by $X_{S,T}$. We wish to show that with positive probability, none of the bad events occur. *(Try to convince yourself the case $t = \frac{5d}{8}s$ suffices, rather than $t \leq \frac{5d}{8}s$.)*

In total, there are sd edges leaving S. We study the probability of one bad event $X_{S,T}$ as follows:

- Interpret the procedure of sampling d vertices without replacement as follows: First pick a vertex uniformly at random from m options (in R), then another uniformly at random from the remaining $m-1$ options, and so on, down to $m-d+1$ options on the d-th selection.
- First consider a single vertex $u \in s$. By the preceding interpretation, the probability that all of its neighbors are in T is given by

$$\frac{t}{m} \cdot \frac{t-1}{m-1} \cdot \ldots \cdot \frac{t-d+1}{m-d+1} \leq \left(\frac{t}{m}\right)^d,$$

where the inequality follows from $\frac{A-1}{B-1} \leq \frac{A}{B}$ when $A \leq B$ (e.g., $\frac{2}{3} \leq \frac{3}{4}$), and we have $t \leq m$ because $T \subseteq R$. (Note also that $t = \frac{5d}{8}s \leq \frac{5d}{8}\frac{n}{10d} = \frac{n}{16}$, whereas we assume $m \geq 3n/4$.)

- Since the random neighbors selected for each $u \in S$ are independent of each other, it follows that the overall probability of $X_{S,T}$ is at most $(t/m)^{sd}$.

For the desired expansion property to hold, we require that $X_{S,T}$ is false for all possible choices of S and T. Thus, the probability is *failing* to get the desired expansion condition is

$$\begin{aligned}
\Pr\left[\bigcup_{S,T}\{X_{S,T}\}\right] &\overset{(a)}{\leq} \sum_{S,T}\Pr\left[X_{S,T}\right] \\
&\overset{(b)}{\leq} \sum_{S,T}(t/m)^{sd} \\
&\overset{(c)}{\leq} \sum_{s=1}^{n/10d}\binom{n}{s}\binom{m}{5ds/8}\left(\frac{5ds}{8m}\right)^{sd} \\
&\overset{(d)}{\leq} \sum_{s=1}^{n/10d}\left(\frac{ne}{s}\right)^{s}\left(\frac{8me}{5ds}\right)^{5ds/8}\left(\frac{5ds}{8m}\right)^{sd} \\
&\overset{(e)}{\leq} \sum_{s=1}^{n/10d}\left(\frac{1}{20}\right)^{s} \\
&\leq \sum_{s=1}^{\infty}\left(\frac{1}{20}\right)^{s} \\
&= \frac{1}{19},
\end{aligned} \tag{9}$$

where (a) uses the union bound, (b) was shown in the dot points above, (c) follows by counting the number of S and T with $|S| = s \leq \frac{n}{10d}$ and $|T| = t = \frac{5d}{8}s$, (d) uses $\binom{n}{k} \leq (ne/k)^k$, and (e) is a nuisance to work out line-by-line but essentially amounts to showing that each summand in the previous line is small (at most $\left(\frac{1}{20}\right)^s$) under the assumed conditions $d \geq 32$, $m \geq 3n/4$, and $s \leq \frac{n}{10d}$. The details:

- Define $a = \frac{ne}{s} \cdot \left(\frac{8me}{5ds}\right)^{5d/8} \cdot \left(\frac{5ds}{8m}\right)^{d}$, so that the summands in (d) are a^s.
- The dependence on s is $\frac{1}{s} \cdot s^{d(1-5/8)}$, so since $d \geq 32$, it is increasing in s. Thus, we may upper bound a by replacing s by its upper bound $\frac{n}{10d}$ to get $a \leq 10de\left(\frac{16me}{n}\right)^{5d/8}\left(\frac{n}{16m}\right)^{d}$.
- Now the dependence on m is $m^{d(5/8-1)}$, which is decreasing, so we can get an upper bound by replacing m by its smallest value $\frac{3n}{4}$: $a \leq 10de \cdot (12e)^{5d/8} \cdot \left(\frac{1}{12}\right)^{d}$.
- A direct calculation gives $(12e)^{5/8} \times \frac{1}{12} < 0.736$, giving $a \leq 10de \cdot (0.736)^d$, which we can directly verify (numerically or analytically) to be below $\frac{1}{20}$ when $d \geq 32$.

For a more general analysis using generic constants instead of $\frac{5}{8}$, $\frac{1}{20}$, etc., refer to Ola Svensson's notes[4].

Since there is at most a $\frac{1}{19}$ chance of at least one bad event occurring, there is at least an $\frac{18}{19}$ chance of none of them occurring, as desired. □

[4] https://theory.epfl.ch/courses/topicstcs/Lecture3.pdf

3.1 Bipartite expander graphs from general expander graphs

While we have just proved the existence of bipartite expanders using the probabilistic method, it is also useful to note another general approach to getting bipartite expanders. Specifically, we describe an alternative approach in which a regular graph can be "converted" into a bipartite graph while maintaining its expansion properties, which will generally be easier than performing an entirely separate analysis/construction.

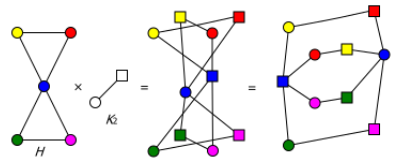


Figure 2: Double cover for a graph

The idea is a notion called the *double cover* of a (non-bipartite) graph G, which is a bipartite graph H with twice the number of vertices and edges as G. To construct H, duplicate vertices in graph G such that for each vertex v_i of G, graph H has two corresponding vertices u_i and w_i. Next, for each edge in G, connect the corresponding vertices across the 2 halves of H, i.e. for an edge (v_i, v_j) in G, replace the edge in H with the edges (u_i, w_j) and (w_i, u_j). Mathematically, for a graph with adjacency matrix A, the adjacency matrix of the double cover is

$$\begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix}.$$

The resulting bipartite graph H remains d-regular. See Figure 2 for a simple example.

Observe that when the original graph satisfies $|N(S)| \geq \alpha|S|$, so does the converted bipartite graph. This is because for each vertex and its neighbors in the original graph, the vertex is connected to the same corresponding vertices in the other mirrored half of the graph after the double cover operation. Hence, good expansion of G implies good expansion of H.

4 Explicit Constructions

There are two ways to define the explicit construction of expander graphs.

Definition of explicitness:. A deterministic algorithm outputs the expander graph's entire adjacency matrix in poly(n) time.
Definition of strong explicitness. Given any $u \in [n]$ (a vertex index), $i \in [d]$ (a neighbor index of that vertex), a deterministic algorithm outputs the i-th neighbor of u in poly(log(n)) time.

Observe that in the latter definition, we are not asking for the full adjacency matrix to be formed, but rather, for specific entries of it to be computable *very efficiently* (poly-log instead of poly). As we will see below, this allows working with graphs with en extremely large number of vertices (e.g., 2^b for b that may be in the hundreds or more), in which case we would certainly like to avoid constructing the full adjacency matrix.

Of course, strong explicitness implies explictness, because one can just loop over all $n \times n$ entries of the adjacency matrix and compute them one-by-one, incurring a total time of $O(n^2 \text{poly}(\log n))$. On the other hand, explicitness does not imply strong explicitness.

The following subsections give a few well-known (strongly) explicit constructions of expander graphs; their expansion properties will be stated without proof. The theorems all state a spectral expansion property,

but these can be related to edge expansion via (6), and to vertex expansion via other tools that we didn't cover (other than a very crude one with a factor of d). These examples follow the ones in the CMU lecture https://www.youtube.com/watch?v=j6JzqPkvRHM.

4.1 (**Optional**) Margulis-Gabber-Galil Expanders

Let $\mathbb{Z}_m = \{0, 1, \ldots, m-1\}$, and consider mod-$m$ arithmetic over this set. Let $V = \mathbb{Z}_m^2$, where vertices are indexed by $x, y \in \mathbb{Z}_m$ on a two dimensional grid with $m \in \mathbf{Z}^+$ points in each dimension (so m^2 total). Construct the edge set E by connecting each vertex with coordinate (x, y) to the following 8 neighbors:

1. $(x \pm y, y)$
2. $(x \pm (y+1), y)$
3. $(x, y \pm x)$
4. $(x, y \pm (x+1))$

where the $+$ and $-$ operators are performed modulo m [Goldreich, 2011]. Note that this could mean connecting a vertex to itself, or connecting one vertex to another multiple times, meaning it is technically a *multi-graph*, but we won't worry so much about this distinction.

Theorem 4.1. *[Gabber and Galil, 1981] The method described above constructs a $(m^2, 8, \epsilon)$-spectral expander graph for some $\epsilon > 0$ ($\epsilon \approx 0.1$)*

This method is strongly explicit, since finding the i-th neighbor of any vertex can trivially be done in $O(1)$ time. Beyond the fact that it is strongly explicit, we see that it is remarkably simple, being completely described by just 4 extremely basic equations.

The proof of Theorem 4.1 is linear algebra based and is less straightforward.

4.2 (**Optional**) Ramanujan graph expanders

The constant $\epsilon \approx 0.1$ in Theorem 4.1 is reasonable, but an analysis of random graphs suggests we can do much better – a random d-regular graph will in fact give a Laplacian matrix L with eigenvalues (except the one that is zero) very close to one, namely $1 - O\left(\frac{1}{\sqrt{d}}\right)$, suggesting that we could get spectral expansion with $\epsilon \approx 1$.

In this section, we introduce another strongly explicit construction called *Ramanujan graph expanders* that can attain an analogous improvement for certain d values. Since we are interested in eigenvalues of L that are close to one, it is more convenient to work with one minus the eigenvalues of L as follows.

Definition 5. *A Ramanujan graph is a d-regular graph that satisfies*

$$\kappa := \max\{|\kappa_i| : i \in [n-1]\} \leq \frac{2}{\sqrt{d}} \sqrt{1 - \frac{1}{d}} \tag{10}$$

where κ_i are the eigenvalues for the normalised adjacency matrix $K = \frac{1}{d}A$. (Since the normalized Laplacian matrix is $L = I - K$ and has eigenvalues λ_i, we have the relationship $\kappa_i = 1 - \lambda_i$.)

Observe that when d is large, such a graph is an expander (from the definition of spectral expansion in Definition 3), and moreover, the expansion constant ϵ is very close to one. Remarkably, the closeness to one not only matches what random graphs give, but does so in a cleaner non-asymptotic manner.

Theorem 4.2. *There exists a strongly explicit construction of a d-regular Ramanujan graph for any $d \geq 3$ of the form $d = p^k + 1$, where $k \in \mathbb{Z}^+$ and p is a prime number.*[5]

The constructions themselves (which we haven't described) are not especially complicated, but the analysis of the resulting eigenvalues are very advanced based on tools from number theory. While the expansion properties are strongest for large d as mentioned above, the special case of $d = 3$ comes out to be particularly simple as follows.

Corollary 4.3. *Let $V = \mathbb{Z}_n$ with n being a prime number, and let E be the set of edges in which we connect $a \in V$ to $a + 1$, $a - 1$ and a^{-1} in the finite field based on mod-n arithmetic (take 0^{-1} to produce 0). Then this is a $(n, 3, \epsilon)$-spectral expander with $\epsilon \approx 0.01$.*

Intuitively, the edges $(a, a + 1)$ and $(a, a - 1)$ produce a cycle of all the vertices $a \in V$, while the edges (a, a^{-1}) produce "pseudo-random edges" that improve connectivity. As a result, the resultant graph from this construction achieves spectral expansion with similar behavior as a random 3-regular graph.

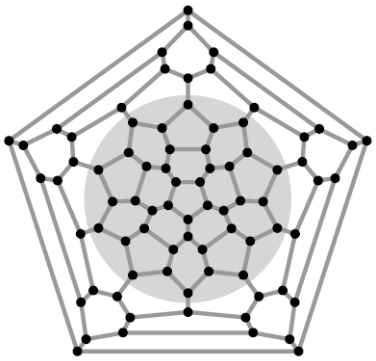


Figure 3: Ramanujan expander graph with 80 vertices [Sarnak, 2004]

4.3 (**Optional**) Zig-Zag product expanders

The final explicit construction that we cover is roughly based on iteratively constructing larger and larger expander graphs. This general approach is especially suited to getting very good bipartite expander graphs, e.g., with the constant ϵ in $|N(S)| \geq \epsilon d|S|$ being a "good" constant like 0.8 rather than an "OK" one like 0.05. The specific method that we cover is simpler and perhaps not quite good enough to attain such constants, but it is easier to understand.

We start with the following definition, which we will typically use with G being a "large" graph and H being a "small" one.

Definition 6. *Given two graphs $G = (V_G, E_G)$ and $H = (V_H, E_H)$ where G is an n-vertex, D-regular graph and H is a D-vertex, d-regular graph. We define the **replacement product**, $G(\mathrm{r})H$, as a $2d$-regular graph with a vertex set $V_G \times V_H$ in which each vertex in G is replaced by a copy of H and (g, h) has an edge to (g', h') if and only if either (i) $g = g'$ and $(h, h') \in E_H$, or (ii) $g \neq g'$, g' is the h-th neighbor of g in G, and g is the h'-th neighbor of g' in G.*

This definition is a bit complicated, but should become clearer with an example shown below:

[5] Early works in this direction further assumed that $p \equiv 1 \pmod 4$, but that assumption was subsequently dropped.

- $G = G_1$ has $n = 7$ nodes and degree $D = 6$;
- $H = G_2$ has $D = 6$ nodes and degree $d = 2$;
- The replacement product $G_1 \textcircled{r} G_2$ contains n "copies" of G_2, and there are further edges connecting different copies in a manner that matches the neighbor numberings in the original G_1.

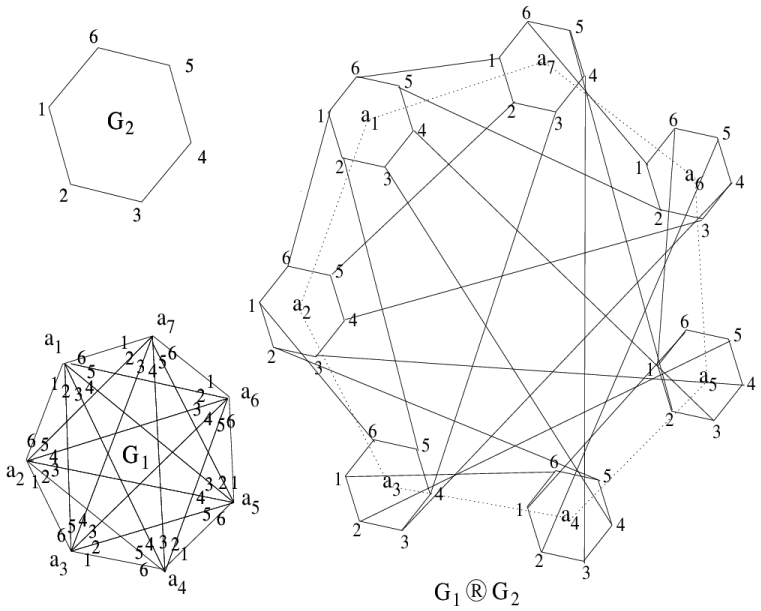


Figure 4: Replacement product example from the paper "Zig-zag and replacement product graphs and LDPC codes" (Kelley/Sridhara/Rosenthal, 2006).

As it turns out, if G and H are both spectral expanders, then so is their replacement product.

Theorem 4.4. *Consider the case that G is a (n, D, ϵ_G)-spectral expander and H is a (D, d, ϵ_H)-spectral expander. Then, the replacement product $G \textcircled{r} H$ is a $(Dn, 2d, \frac{\epsilon_H \epsilon_G}{16})$-spectral expander.*

Notice that the number of vertices has increased (the sizes of G and H get multiplied), the degree has changed from D to $2d$ which is typically a decrease (due to G being larger than H), and the spectral expansion constant has decreased. The last of these is not desirable, but there is a way to increase the expansion factor. We only provide a brief overview of the idea as follows.

Given the graph G, consider G^t, which is a new graph with edges added based on the t-step connectivity (if there exists a t-step path from edges u to v in G, add an edge to them). Then G_t has a degree of d^t (including multi-edges). It can be shown that after performing this idea on the replacement product, the resulting graph is "approximately" a $(Dn, (2d)^t, t\frac{\epsilon_H \epsilon_G}{16})$-expander (this statement is a bit imprecise, as the expansion factor is only an approximate expression).

By carefully interleaving the two steps (replacement product and $(\cdot)^t$), we can get a good expander graph – the replacement product helps keep the degree from getting too large, and the $(\cdot)^t$ step prevents the spectral expansion parameter from getting too small. See the CMU video lecture for somewhat more detail (though it is still on the brief side).

5 Applications

5.1 Error correcting codes

Consider the case where a sender must send a message to a receiver over a noisy channel that can flip an arbitrary fraction of the k bits of the original message. Knowing that the channel is noisy, the sender adds redundancy by encoding the message into $n > k$ bits. Any such resulting length-n string is called a *codeword*. The hope is that this can be done in a manner that attains multiple goals:

- Send information at a high rate (i.e., keep $\frac{k}{n}$ as high as possible);
- Achieve resilience to errors (i.e., correct decoding is guaranteed even when there are δn bit flips; the higher δ the better);
- Maintain low computational complexity at the encoder and decoder.

We will show how to use expander graphs to build such a code.

The notions of high rate and resilience to errors are formalized as follows.

Definition 7. ***(Code rate)***. *The rate of a code $C \subset \{0,1\}^n$ (with $|C| = 2^k$) is $R = \frac{k}{n}$.*

Definition 8. ***(Minimum distance)***. *The normalized minimum distance (or "distance" for short) of a code $C \subset \{0,1\}^n$ is $D = \min_{c_1 \neq c_2} \frac{1}{n} d_H(c_1, c_2)$, where d_H is the Hamming distance.*

The higher the D, the more tolerant is our coding method to errors during transmission. In particular, it can be shown that an optimal decoder is always able to uniquely recover the message when there are $\frac{D-1}{2}$ or fewer bit flips.

In general, when the rate is high, the distance tends to be small (and vice versa). A sequence of codes (indexed by k) is said to be *asymptotically good* the rate and distance are both lower bounded by positive constants as $k \to \infty$ (e.g., $R \geq 0.01$ and $D \geq 0.01$).

- Note: Even better would be to get the rate and distance both as high as possible (instead of just "any constant value"), and expanders can be good for that too, as well as being extremely efficient computationally (in particular, $O(n)$ decoding time is attainable). However, we'll only focus on this more modest goal here, and we won't place much emphasis on computation.

We can obtain a asymptotically good code using bipartite expanders. We give a specific example using specific constants, but this can be generalized to get a more general trade-off between rate and distance.

To represent the error correcting code as a bipartite expander, we consider codewords of length n, with $m = \frac{3}{4}n$ parity check constraints. The dimension of the code is $k = n - m = \frac{1}{4}n$, representing a constant rate of $\frac{1}{4}$. We represent the error correcting code as a bipartite graph with $|L| = n$ and $|R| = m$. Such a bipartite graph is called the *Tanner graph* of C. The vertices in the left subgraph L represent codeword bits, while the vertices in the right subgraph R represent parity check equations that all codewords must satisfy (and such that if all of them are satisfied, we must have a codeword). For each parity check, the bits connected to it are required to consist of an even number of 1s. It turns out to work well to let this graph be a bipartite expander graph, and we will specifically adopt the choice from Theorem 3.1, whose main property we repeat here for convenience:

$$|N(S)| \geq \frac{5d}{8}|S|, \quad \forall S \subseteq L : |S| \leq \frac{n}{10d}. \tag{11}$$

Consider the adjacency matrix of the bipartite expander graph $\boldsymbol{H} \in \{0,1\}^{n \times m}$, where $H_{i,j}$ (with $i \in L$ and $j \in R$) is the indicator for when there is an edge from vertex i in the left subgraph to vertex j in the right subgraph. Then, a string $\boldsymbol{x} \in \{0,1\}^n$ is a valid codeword if and only if all the parity checks are satisfied:

$$\boldsymbol{x} \in C \iff \bigoplus_{i=1}^{n} H_{i,j} x_i = 0, \ \forall j \in \{1 \cdots m\} \tag{12}$$

where $\oplus$ is mod-2 addition. We make use of the properties of the bipartite expander graph (from Theorem 3.1, repeated in (11)) to analyse the Hamming distance of the code C.

Lemma 5.1. *Consider the bipartite expander graph from Theorem 3.1. For any subset $S \subseteq L$ with $|S| \leq \frac{n}{10d}$, there exists a vertex $v \in N(S)$ with exactly one neighbor in S.*

Proof. Assume, for the sake of contradiction that for all $v \in N(S)$, it holds that $|N(v) \cap S| \geq 2$. Then,

$$(\#\text{edges from } S \text{ to } N(S)) \geq 2|N(S)| \geq 2 \cdot \frac{5d}{8}|S| > d|S|,$$

where the $\frac{5d}{8}$ term comes from the expansion property in Theorem 3.1. This contradicts with the fact that the graph is d left regular, meaning there are only $d|S|$ edges containing vertices from S. □

Corollary 5.2. *The minimum Hamming distance of the code C is greater than $\frac{n}{10d}$, which is linear in n.*

Proof. Let $\boldsymbol{x}$ be any valid codeword, and let $\boldsymbol{x'}$ be any other sequence whose Hamming distance to $\boldsymbol{x}$ is at most $\frac{n}{10d}$. Let S be the set of indices at which $\boldsymbol{x}$ and $\boldsymbol{x'}$ differ. From Lemma 5.1, there exists some $v_i \in N(S)$ with exactly one neighbor in S. This single-neighbor property translates to the following: *For the i-th parity check, one of its bits is different in $\boldsymbol{x}$ and $\boldsymbol{x'}$, and all of its other bits are identical in $\boldsymbol{x}$ and $\boldsymbol{x'}$.*

Since $\boldsymbol{xH} = \boldsymbol{0}$ (due to $\boldsymbol{x}$ being a codeword), the property just stated implies that the i-th entry of $\boldsymbol{x'H}$ is 1, meaning $\boldsymbol{x'}$ is not a codeword.

Thus, for any codeword $\boldsymbol{x}$, there are no other codewords within Hamming distance $\frac{n}{10d}$. □

Next, we proceed to consider the decoding step, which seeks to recover $\boldsymbol{x}$ from its corrupted version $\boldsymbol{y} = \boldsymbol{x} \oplus \boldsymbol{z}$ (with $\boldsymbol{z} \in \{0,1\}^n$ containing a 1 wherever a bit is flipped).[6] We will focus only on a very simple decoder, described as follows.

Input: $\boldsymbol{y}$ such that $\boldsymbol{yH} \neq \boldsymbol{0} \pmod 2$
Output: An estimated codeword $\hat{\boldsymbol{x}}$ (desired to be as close to $\boldsymbol{y}$ as possible)

1: $\hat{\boldsymbol{x}} \leftarrow \boldsymbol{y}$
2: **while** $\hat{\boldsymbol{x}}\boldsymbol{H} \neq \boldsymbol{0} \pmod 2$ **do**
3: Flip any $\hat{x}_i$ that decreases the number of constraint violations in $\hat{\boldsymbol{x}}\boldsymbol{H}$

We omit a full analysis of this algorithm and only outline some of the main ideas:

- Suppose that we are given some $\boldsymbol{y}$ withing distance $\frac{n}{10d}$ of the correct codeword (i.e., at most $\frac{n}{10d}$ bit flips occurred).
- Using similar reasoning as the proof for Corollary 5.2, at each iteration of the loop, there exists at least one constraint violation associated with exactly one bit in $\hat{x}_i$ ($i \in S$), where S is the index set of parity checks that are violated. Flipping that bit would reduce the number of constraint violations by one.

[6] Actually the goal is to recover the original message bits, but that's straightforward once $\boldsymbol{x}$ is recovered.

- There may be better choices that reduce by more than one, but even so, we can conclude that the number of constraint violations will *strictly decrease* on each iteration. Thus, this number will eventually decrease to zero, meaning we end up with a valid codeword.

The preceding argument does not establish that we will end up with the *closest* codeword, but this turns out to also be true. For the proof of that, see Lecture 8 by Venkatesan Guruswami[7], which also describes an improved decoding algorithms with a *linear* runtime of $O(n)$. (The above algorithm has polynomial runtime, but not linear.)

5.2 Error reduction in randomized algorithms

Expanders are also useful in area of error reduction (and derandomization more broadly). The goal in this problem is to reduce the error probability of a randomized algorithm without using too many extra random bits. The motivation is that random bits are often viewed as a scarce resource, so the fewer that are needed, the better. Morever, if we bring it down to a small enough number k, we can even get a deterministic algorithm by just searching over all 2^k random seeds.

Let $\mathcal{A}$ be a randomized algorithm for solving a decision problem (i.e., something with a YES/NO answer). Suppose that we require a *one-sided* error guarantee:

- If the correct answer is YES, the output must be YES with probabiltiy one;
- If the correct answer is NO, the output must be NO with some specified probability (e.g., 2/3 or 0.95).

For example, the algorithm might be for checking whether an input number is prime: If the number is prime, $\mathcal{A}$ returns 1 with probability 1, and if the number is non-prime, $\mathcal{A}$ returns 1 with probability ≤ 0.05 - for example, the Miller-Rabin primality test has such properties.

Assume that this algorithm makes use of a random n-bit string from $\{0,1\}^n$ and makes a mistake over at most a fraction 0.05 (say) of all n-bit random strings.

5.2.1 Naive approach

One naive approach for reducing the error of our random algorithm is to repeat it d times with a different random n-bit string each time. This incurs:

1. dn random bits.
2. at most 0.05^d chance of being wrong. (Hence, we need $d = O\big(\log \frac{1}{\delta}\big)$ to get down to some target $\delta < 0.05$.)
3. algorithm runtime of $dT + O(n)$ (including $O(n)$ time for generating the random bits), where T is the time for a single invocation.

This approach requires us to regenerate a new random n-bit string on each invocation of the algorithm, and it turns out that this can be avoided/alleviated using expanders.

[7] https://www.cs.cmu.edu/ venkatg/teaching/codingtheory/notes/notes8.pdf

5.2.2 Improved approach using expanders

Suppose that we have a strongly explicit algorithm to generate a bipartite expander that has the properties in Theorem 3.1 with $|L| = |R| = 2^n$. For both L and R, we represent any given vertex via a unique n-bit string, for a total of 2^n vertices on each side. We will interpret these strings as choices of the "random seed" for the randomized algorithm.

Now, consider the algorithm described as follows. First pick a random vertex ℓ from L; this is equivalent to picking a n-bit random string (just like in the naive approach). Next, use the strongly explicit algorithm to generate the d neighboring vertices of $\ell : r_1, r_2, \ldots, r_d$ that form $N(\ell) \subseteq R$ in the bipartite expander. Due to the strongly explicit property, this takes $\text{poly}(\log(2^n)) = \text{poly}(n)$ time. We now have d n-bit strings which we run $\mathcal{A}$ with, and we return 1 if the answer is positive for all d trials, and 0 otherwise .

Claim: The preceding algorithm incurs:

1. n random bits.
2. at most $\frac{0.1}{d}$ chance of being wrong (proved below).
3. a runtime of $dT + \text{poly}(n)$, where T is the time for a single invocation.

Notice that compared to just running $\mathcal{A}$ once, the algorithm uses the *same* number of random bits but has a smaller error rate ($\frac{0.1}{d} \leq 0.05$ for $d \geq 2$), albeit at the cost of a higher runtime (at least a factor d larger).

Proof of error rate. Since the algorithm makes use of n-bit strings on R (right partition) of our bipartite expander, let $B_x \subseteq R$ denote "bad" n-bit strings which cause $\mathcal{A}$ to give the wrong answer. By the assumption of having error probability at most 0.05, we have that B_x consists of at most a 0.05 fraction of the number of n-bit strings i.e., $|B_x| \leq 0.05(2^n)$. Correspondingly, define $S \subseteq L$ to contain the "bad" choices in L such that if we choose any $\ell \subseteq S$ in the algorithm, we will have all of its neighbors in B_x (i.e., $N(\ell) \subseteq B_x$ – this is the only case where the algorithm makes an error; if any vertex in $N(\ell)$ lies outside of B_x, then the algorithm will output the correct answer). The key observation is stated in the following lemma.

Lemma 5.3. *For $d \geq 32$, the number of bad choices $|S|$ in L is such that $|S| < \frac{0.1}{d} 2^n$*

Proof. For the sake of contradiction, assume that $|S| \geq \frac{0.1}{d} 2^n$. Then, consider $S' \subseteq S$ such that $|S'| = \frac{0.1}{d} 2^n$. The bipartite expansion property from Theorem 3.1 (which assumes $d \geq 32$) gives the following:

$$|N(S')| \geq \frac{5}{8} d |S'| \geq \frac{5}{8} d \frac{0.1}{d} 2^n = \frac{1}{16}(2^n) > |B_x|, \tag{13}$$

where the last step follows by recalling that $|B_x| \leq 0.05(2^n)$. This implies that there exist some choices in S' such that the algorithm does not produce the wrong answer (since the algorithm will only give a wrong answer if all vertices chosen in R are in B_x). This leads to a contradiction, because S' can only contain "bad" choices. Thus, our original assumption $|S| \geq \frac{0.1}{d} 2^n$ must have been incorrect, meaning we indeed have $|S| < \frac{0.1}{d} 2^n$. □

Therefore, the probability of us picking one of these "bad" choices in L is at most $\frac{|S|}{2^n} < \frac{0.1}{d}$, which completes the claim.

Overall, the expansion property is a useful sufficient condition for ensuring that this error reduction

technique works well. In principle we could use any bipartite graph, but graphs with poor expansion properties may be significantly less useful. This is because vertices in L might map to many identical vertices in R, which limits what can be gained by running all the resulting random seeds. The properties of expanders prevent such undesirable scenarios.

5.3 (**Optional**) Other applications

We briefly note that expanders also arise in many other topics in theoretical computer science and applied domains. Some examples and corresponding paper titles are as follows:

- Cryptography (e.g., "Cryptographic hash functions from expander graphs")
- Sparse estimation (e.g., "Efficient and robust compressed sensing using high-quality expander graphs")
- Group testing (e.g., "Derandomization and group testing")
- Circuit complexity lower bounds (e..g, "Poly-logarithmic Frege depth lower bounds via an expander switching lemma")
- Network design (e.g., "Optimal network topologies: Expanders, cages, Ramanujan graphs, entangled networks and all that")
- Deep neural networks (e.g., "Deep expander networks: Efficient deep networks from graph theory")

References

Ofer Gabber and Zvi Galil. Explicit constructions of linear-sized superconcentrators. *Journal of Computer and System Sciences*, 22(3):407–420, 1981.

Oded Goldreich. *Basic Facts About Expander Graphs*, pages 451–464. Springer Berlin Heidelberg, 2011.

K.P. Hart, J. van Mill, and P. Simon. *Recent Progress in General Topology III*. SpringerLink : Bücher. Atlantis Press, 2013.

Peter Clive Sarnak. What is . . . an expander? *Notices of the American Mathematical Society*, 51(7):762–763, August 2004.

CS5275 Lecture 11: Communication Complexity

Jonathan Scarlett

Acknowledgment. The first version of these notes was prepared by Yao Tong and T.-Duy Nguyen-Hien for a CS6235 assignment.

Useful references:

- Blog post: The complexity of communication (Math ∩ Programming)
- TCS toolkit style videos:
  - Basics: `https://www.youtube.com/watch?v=mQQ36cDnmR8`
  - Deterministic CC: `https://www.youtube.com/watch?v=zFHWmmThdT4`
  - Randomized CC: `https://www.youtube.com/watch?v=LRef5a88uZQ`
- Lecture notes
  - CMU CS15-859T
  - Rutgers CS514
  - UCSD CSE291
- Textbooks:
  - A. Rao and A. Yehudayoff, *Communication Complexity and Applications*. Cambridge University Press, 2020
  - A. A. Razborov, "Communication complexity," in *An Invitation to Mathematics: From Competitions to Research*. Springer, 2011, pp. 97–117
  - T. Roughgarden *et al.*, *Communication complexity (for algorithm designers)*. Now Publishers, Inc., 2016, vol. 11, no. 3–4
  - T. Lee, A. Shraibman *et al.*, "Lower bounds in communication complexity," *Foundations and Trends® in Theoretical Computer Science*, vol. 3, no. 4, pp. 263–399, 2009

Categorization of material:

- Core material: Sections 1–4
- Extra material: Section 5 (Randomized CC), Section 6 (Applications)

(Exam will strongly focus on "Core". Take-home assessments may occasionally require consulting "Extra".)

1 Overview

Communication complexity was introduced by Yao in the late 1970s [5], and studies the minimum number of bits that two (or more) parties must exchange in order to cooperatively accomplish some task based on their inputs. This is abstracted by the problem of *computing a function* $f(x, y)$, where one party knows x and the other knows y.

Communication complexity has some conceptual similarity and even technical overlap with the communication problem we mentioned in the information theory lecture, but with some major differences:

- Most importantly, the goal is to compute a function, not to send a message.
- The communication is typically two-way rather than one-way.
- We usually consider the number of bits with respect to the "worst-case" x and y (possibly restricted to lie in certain sets), rather than putting probabilistic models on them.
- We usually consider the noiseless exchange of bits, rather than noisy communication channels.
- We usually settle on understanding scaling laws in terms of big-O notation (e.g., $O(n)$ vs. $O(\log n)$ when x, y are in $\{0,1\}^n$), rather than trying to establish the precise constants.

Having said this, information theory *does* play a major role in several proofs on communication complexity, particularly for randomized protocols (such proofs will be beyond the scope of this lecture).

Perhaps the main use of communication complexity is not to directly model the real world, but rather, to provide a useful abstraction that connects to extensive other fields throughout theoretical computer science. In particular, when a problem in communication complexity is known to be "hard" in the sense of requiring a large number of bits to be exchanged, reductions can be used to show other problems to be hard, e.g.:

- Lower bounds on the storage required in streaming algorithms;
- Lower bounds on query complexity in learning problems;
- Lower bounds on sample complexity in data-centric problems.

We will give some examples in Section 6.

Deterministic vs. randomized protocols. One of the key defining features of communication complexity problems is *determinsitic* vs. *randomized*. For deterministic protocols, both parties are required to act deterministically and produce the correct output. For randomized protocols, the parties by make random decisions, and some probability of error is allowed (e.g., output the correct answer with probability 0.9). This distinction can have a major impact on the amount of communication required, e.g. $O(n)$ deterministic vs. $O(\log n)$ randomized (as we will see later).

Generally speaking, randomized communication complexity turns out to be more useful when it comes to forming reductions to other problems in computer science. However, deterministic communication complexity is much more suitable and digestible for an introductory lecture on the topic. Thus, we will focus mainly on the deterministic case in Sections 2–4, then briefly look at the randomized case in Section 5, before giving examples of reductions (for both deterministic and randomized settings) in Section 6.

2 Problem Setup (for Deterministic Protocols)

Assume there are two parties, Alice and Bob, with Alice having access to an input $x \in \mathcal{X}$ and Bob having access to another input $y \in \mathcal{Y}$. Their collective objective is to compute the function value $f(x, y)$, where the function $f : \mathcal{X} \times \mathcal{Y} \to \mathcal{Z}$ is mutually known. The main obstacle is that the two parties are unaware of each other's inputs (*i.e.*, Alice knows *only* x and Bob knows *only* y). Thus, to compute the value $f(x, y)$, they will need to communicate with each other, following a fixed protocol agreed upon beforehand.

- Note: Unless stated otherwise, we will take $\mathcal{X} = \{0, 1\}^n$ and $\mathcal{Y} = \{0, 1\}^n$ (i.e., the inputs are length-n bit strings), and $\mathcal{Z} = \{0, 1\}$ (i.e., the output is binary), which is the most natural starting point.
- Note: Since we are specifically interested in communication, we allow Alice and Bob to individually have unlimited storage and computation (though keeping these low is still considered preferable).

A **deterministic communication protocol** is a protocol π that determines which player sends the next message and what they send to the other player. Each message sent may depend on the player's own knowledge (including their input x or y) and the *communication history* (all the previously communicated bits up to that point). It is common for the final message transmitted to represent the "final output" $\pi(x, y)$, which equals $f(x, y)$ if the protocol successfully computes f. However, we *do not strictly require that to be the case.*[1] Instead, we consider the following more general notion.

Definition. *We say that a protocol π computes a function f if, for all x and y, after communication ends, the value $f(x, y)$ can deterministically be computed by both Alice and Bob given the information they received during the protocol (and their own input).*

For any given function f, there will exist multiple protocols achieving this goal. As already hinted, we specifically interested in protocols that require as little communication as possible.

Definition. *The communication cost of a protocol is the total maximum number of bits exchanged by the two parties, where the maximum is taken with respect to all possible input pairs (x, y). The communication complexity of a function f is the smallest such communication cost among all protocols.*

We start with some simple examples to better understand these terms.

2.1 Example 1: OR Function

Assume Alice and Bob are two parties who want to compute the OR function of their binary inputs without revealing their entire inputs to each other. Let x be a binary string of length n held by Alice and y be another binary string of length n held by Bob. The function $f : \{0, 1\}^n \times \{0, 1\}^n \to \{0, 1\}$ they want to compute is

$$\mathrm{OR}(x, y) = x_1 \vee y_1 \vee x_2 \vee y_2 \vee \ldots \vee x_n \vee y_n,$$

where x_i and y_i are the individual bits of x and y respectively, and $\vee$ represents the logical OR.

[1] Note, however, that requiring this would only cost at most 1 extra bit if the output of f is binary.

Protocol 1 A naive protocol would be that Alice sends her entire string x to Bob. Then Bob computes the OR function directly and sends the result back to Alice. Hence, the *cost* of this protocol is $n+1$ bits.

In fact, for arbitrary $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$, there is always a "trivial" protocol for f that uses $n+1$ rounds. Alice simply sends her entire n-bit x to Bob. Since Bob knows x, he can send $f(x,y)$ in the next round. Formally, defining $\mathrm{cc}(f)$ as the communication complexity of f, we have the following.

Proposition 2.1. *For every* $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$*, we have* $\mathrm{cc}(f) \leq n+1$.

When n is very large, a protocol using $O(n)$ bits of communication is highly undesirable. Is there any better protocol?

Protocol 2 In this OR function example, a better protocol is that Alice firstly computes $g(x) = x_1 \vee x_2 \vee \ldots \vee x_n$ and sends $g(x)$ to Bob. Then, Bob computes $z = y_1 \vee y_2 \vee \ldots \vee y_n \vee g(x)$ and sends z back to Alice. Therefore, the cost of this protocol is just 2 bits.

2.2 Example 2: EQUALS Function

Another important and useful example is the function EQUALS, defined by

$$\begin{aligned}\mathrm{EQUALS}(x,y) &= \begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise} \end{cases} \\ &= \mathbb{1}(x_1 = y_1) \wedge \mathbb{1}(x_2 = y_2) \wedge \ldots \wedge \mathbb{1}(x_n = y_n),\end{aligned}$$

where $\mathbb{1}$ denotes the indicator function, and $\wedge$ is the logical AND operation.

- Note: This is not just a "toy" theoretical example, but is actually very practically motivated – it corresponds to two parties each having their own version of a file, and they would like to communicate to check whether the two versions are identical. For example, they could have become out-of-date due to updates, database read/write errors, etc.

From Proposition 2.1, we know $n+1$ bits of communication suffice, achieved via Alice sending her entire string. A natural question is then: **Is there room to improve the naive $n+1$ bit protocol?** In Section 4, we will demonstrate that $n+1$ is in fact optimal for deterministic protocols, meaning EQUALS is a much "harder" function than OR. To establish this, we will study lower bounds on communication complexity.

- Note: Jumping ahead slightly, this will turn out to be an example where *randomization helps a lot*. In Section 5, we will discuss simple randomized protocols with communication cost $O(\log n)$ or even $O(1)$, depending on the model of randomness used.

With the over-arching goal of answering the question "**For a given function f, what is its communication complexity?**", we proceed to introduce some key concepts.

3 Representations and Properties of Deterministic Protocols

3.1 Protocol Trees

A natural way to represent a general protocol is using a binary tree in which each branch is based on whether a certain bit computed and sent (by Alice or Bob) is 0 or 1. An example is shown at the top of the next

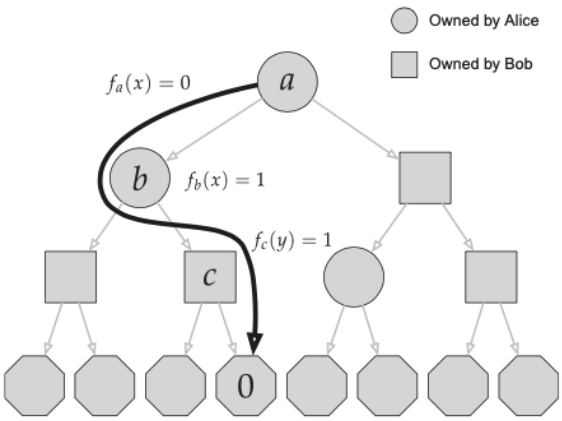


Figure 1: An execution of a protocol in a protocol tree. [1]

page. Elaborating on this example:

- The first bit $f_a(x)$ is computed by Alice and sent to Bob.
- Taking the left branch indicates that $f_a(x) = 0$, leading to the node labeled b. In this case, Alice is responsible for computing and sending another bit $f_b(x)$.
- Taking the right branch indicates that $f_b(x) = 1$, leading to the node labeled c. In this case, Bob is responsible for computing $f_c(y)$ and sending it to Alice.
- Taking the right branch indicates that $f_c(y) = 1$, and at this stage Alice and Bob declare that the protocol output is 0.

This naturally extends to larger trees (i.e., longer protocols). In the above figure, it happens that an output is always decided after sending 3 bits, but in general this could vary depending on which branches are taken.

For any tree, define the *depth* to be the distance (number of branches) of the furthest leaf from the root. With the tree representation, we can immediately deduce the following: **The deterministic communication complexity equals the smallest depth among all protocol trees that compute $f(x, y)$.**

3.2 Rectangles

It is often useful to visualize the function $f(x, y)$ as a matrix with $|\mathcal{X}|$ rows, $|\mathcal{Y}|$ columns, and (x, y)-th entry equal to $f(x, y)$. This will be done both here and when we discuss 'Rank' below.

- **Caution**: If $\mathcal{X} = \mathcal{Y} = \{0, 1\}^n$, then the matrix size is $2^n \times 2^n$, not to be confused with $n \times n$. In this case, the matrix is indexed by *length-n strings*, rather than integers like you might be more used to.

Given a subset $A \subseteq \mathcal{X}$ of the row indices and a subset $B \subseteq \mathcal{Y}$ of column indices, the set $A \times B$ is called a *combinatorial rectangle*; in the following we omit the word "combinatorial" and just call these *rectangles*. Importantly, A and B are not necessarily consecutive indices, so the "rectangle" can have gaps in it, e.g.:

The key reason for focusing on rectangles is that the sharing of bits in a protocol naturally leads to "narrowing down" the location of (x, y) to lie withing rectangles. For example, if Alice tells Bob that $x \in A$ and Bob tells Alice that $y \in B$, then both parties have a mutual understanding that (x, y) lies in the rectangle $A \times B$.

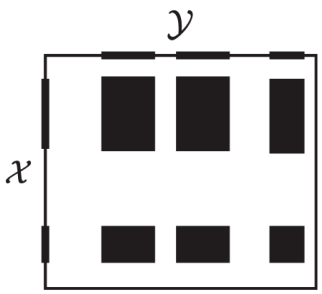


Figure 2: An example of a (combinatorial) rectangle with non-consecutive row and column indcies. [1]

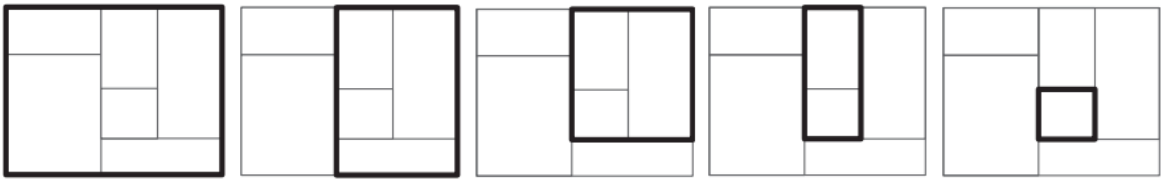


Figure 3: Example of the evolution of a rectangle (taken from [1]). When v is the root node in the protocol tree, X_v is the entire space $\mathcal{X} \times \mathcal{Y}$ (left figure). Each time a bit is communicated by Alice (resp., Bob), the rectangle is partitioned horizontally (resp., vertically), and collectively Alice and Bob both know that (x, y) lies in the rectangle shown.

Taking this idea further, it in fact holds that ***every protocol can be represented using rectangles***. To see this, consider an arbitrary protocol tree, and for each node v, define the subset $S_v \subseteq \mathcal{X} \times \mathcal{Y}$ to contain all input pairs (x, y) that would lead the protocol to node v during its execution, and let

$$X_v = \{x \in \mathcal{X} : \exists y \in \mathcal{Y} \text{ such that } (x, y) \in S_v\}$$
$$Y_v = \{y \in \mathcal{Y} : \exists x \in \mathcal{X} \text{ such that } (x, y) \in S_v\}.$$

The following lemma shows that $S_v = X_v \times Y_v$ (thus being a rectangle), and Figure 3 gives an illustration of this and how branching in the protocol tree amounts to doing a horizontal (Alice) or vertical (Bob) partitioning of the previous rectangle.

Lemma 3.1 (**Lemma 1.4 in [1]**). *For every node v in the protocol tree, the set S_v (defined above) is a rectangle R_v with $R_v = X_v \times Y_v$. Moreover, the rectangles corresponding to the leaves of the protocol tree form a partition of $\mathcal{X} \times \mathcal{Y}$.*

Proof. The lemma follows by induction. For the root node r, we have $R_r = \mathcal{X} \times \mathcal{Y}$, so indeed the lemma holds. Now consider an arbitrary node v such that $R_v = X_v \times Y_v$, and let u, w be the children of v in the protocol tree. Without loss of generality, we can suppose Alice is the one sending a bit at node v, that u is the subsequent node when 0 is sent, and that w is the subsequent node when 1 is sent. In this case, $Y_v = Y_u = Y_w$ since Bob's input is not used in this step. Define:

$$X_u = \{x \in X_v : f_v(x) = 0\}$$
$$X_w = \{x \in X_v : f_v(x) = 1\},$$

where $f_v(x)$ is Alice's sent bit corresponding to node v. It follows that X_u and X_w form a partition of X_v,

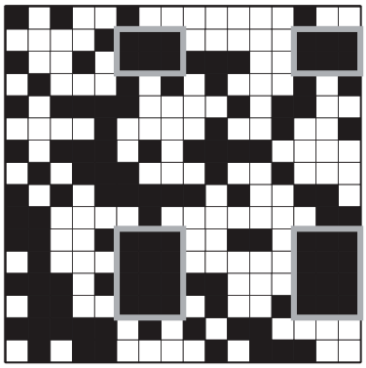
and $R_u = X_u \times Y_u$ and $R_w = X_w \times Y_w$ form a partition of R_v. Having established the base case and the induction step, the desired claim follows for all v, and we deduce that the rectangles corresponding to the leaves form a partition of $\mathcal{X} \times \mathcal{Y}$. □

3.2.1 Monochromatic Rectangles

To utilize rectangles as building blocks, we need to further introduce the concept of *monochromatic rectangles*.

Definition 3.1 (**Monochromatic Rectangles** [1]). *Given $f : \mathcal{X} \times \mathcal{Y} \to Z$ and a constant $z \in Z$, a rectangle R is z-monochromatic if $f(x, y) = z$ for all $(x, y) \in R$.*

The following figure gives an example of a Boolean function $f : \mathcal{X} \times \mathcal{Y} \to \{0, 1\}$ and one (combinatorial) rectangle that is 1-monochromatic (black):



Recalling that each leaf node in the protocol has an associated rectangle, and that reaching the leaf node amounts to outputting a specific value, we immediately deduce the following.

Fact 1. *If a protocol π computes a function $f : \mathcal{X} \times \mathcal{Y} \to \{0, 1\}$ and v is a leaf node in the protocol tree of π, then the rectangle R_v corresponding to node v is a monochromatic rectangle with respect to f.*

Since the number of leaves in a rooted binary tree of depth d is at most 2^d, we deduce the following simple but important result.

Theorem 3.2 (**Theorem 1.6 in** [1]). *If the communication complexity of $f : \mathcal{X} \times \mathcal{Y} \to \{0, 1\}$ is c, then $\mathcal{X} \times \mathcal{Y}$ can be partitioned into at most 2^c monochromatic rectangles with respect to f.*

A particularly useful consequence of this result is the contrapositive statement: **If the function f is such that we cannot form a partition of $\mathcal{X} \times \mathcal{Y}$ using at most 2^c monochromatic rectangles, then the communication complexity must exceed c.** This will be one of the methods for deriving lower bounds on communication complexity in Section 4.

3.2.2 From Rectangles to Protocols

Theorem 3.2 shows that with communication complexity c, we will have at most 2^c monochromatic rectangles. This raises the natural question of whether we can state a similar kind of reverse relationship – if we can form a partition of f using a small number of monochromatic rectangles, can we conclude that the communication complexity is small?

A subtle issue here is that *not every partition of rectangles can be induced by a protocol*; see Figure 4 for a simple counter-example. A protocol always amounts to sequentially dividing a larger rectangle into smaller ones, whereas this "windmill-like" behavior does not fit that description. Nevertheless, it turns out that

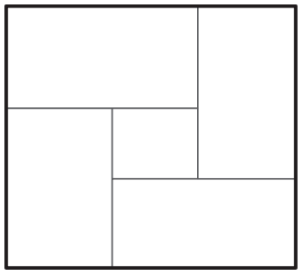


Figure 4: A partition into rectangles that cannot be realized by any protocol. [2]

when there are at most 2^c monochromatic rectangles, we can come up with a protocol of length $O(c^2)$, which is not too much higher than the "ideal" $O(c)$ that would match Theorem 3.2.[2] This is stated as follows.

Theorem 3.3. *Suppose that a function f is such that there exists a partitioning into at most 2^c monochromatic rectangles for some integer $c > 0$. Then there exists a protocol of length $O(c^2)$ that computes f.*

The (optional) proof is given in Appendix B.

3.3 Rank

The third concept we introduce is *rank*, a fundamental notion in linear algebra that can be a useful tool for bounding the communication complexity.

We continue with the notion of representing a function with inputs $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ as an $|\mathcal{X}| \times |\mathcal{Y}|$ matrix, but we now specifically denote that matrix by M (i.e., $M_{xy} = f(x, y)$). We focus mainly on the case that $\mathcal{X} = \mathcal{Y} = \{0,1\}^n$ and binary-valued functions (i.e., each entry of M is 0 or 1). It turns out that the *rank of this matrix* has fundamental connections to communication complexity.

As a reminder, the following fact states some equivalent definitions of matrix rank.

Fact 2. *For any matrix M (not necessarily square), the following are equivalent:*

1. $\text{rank}(M) = r$.
2. *r is the maximum size of a set of linearly independent columns in M. (Or similarly for rows.)*
3. *r is the smallest number such that M can be expressed as $M = AB$, where A is an $m \times r$ matrix, and B is an $r \times n$ matrix.*
4. *r is the smallest number such that M can be expressed as the sum of r matrices of rank 1.*

3.3.1 Communication Complexity and Rank

The following result indicates that *low rank implies low communication complexity*. Later, in Section 4.4, we will further establish that the communication complexity is *lower bounded* by $\log(\text{rank}(M) + 1)$, and discuss further results narrowing the gap between the upper and lower bounds.

[2] Recent work has shown that the square is in fact unavoidable in general [6].

Theorem 3.4 (**Theorem 2.2 in [1]**). *Consider any $f : \mathcal{X} \times \mathcal{Y} \to \{0,1\}$, and let M be the associated binary matrix. Then, the communication complexity of f is at most* $\operatorname{rank}(M) + 1$.

We proceed with the proof. The idea is to use formatulion #3 of rank from Fact 2, meaning that Alice and Bob can employ a factorization $M = AB$, where A is an $m \times r$ matrix and B is an $r \times n$ matrix. Given that Alice has $x \in X$ and Bob has $y \in Y$, let e_x be the standard unit column vector with a 1 in the position corresponding to x, and similarly, let e_y be the one for y. The function can then be computed as follows:

$$f(x, y) = e_x^T M e_y = e_x^T A B e_y.$$

As a result, Alice only needs to send Bob $e_x^T A$, after which Bob multiplies $e_x^T A$ with Be_y and returns the result $M_{xy} = f(x, y)$ to Alice (requiring 1 bit).

As stated, we face the issue that A may be a non-binary matrix, making it unclear how much communication is required to send $e_x^T A$. However, it turns out that we can always ensure that A is binary, meaning we can send $e_x^T A$ using r bits, for a total of $r + 1$ (as desired). This is stated in the following claim.

Claim 1. *It is always possible to factorize a Boolean matrix M by $M = AB$, where A is a binary $m \times r$ matrix and B is a (possibly non-binary) $r \times n$ matrix.*

Proof. Construct A as an $m \times r$ matrix using r columns from M that are linearly independent. Considering that M is a binary matrix, A is a binary matrix as well. Given that r represents the rank of M, every other column in M can be expressed as a linear combination of these r columns. Hence, there exists an $r \times n$ matrix B such that $M = AB$. □

This completes the proof of Theorem 3.4.

4 Lower Bounds for Deterministic Protocols

When studying communication complexity, the natural first step is to devise various protocols, establish their communication cost, and deduce upper bounds on the communication complexity. However, unless we manage to bring the communication cost to something that clearly can't be improved (e.g., $O(1)$ scaling), we might always be left wondering whether we can do better. It turns out that the answer is often "No, we cannot do (much) better", and the way to establish this is by establishing *lower bounds on communication complexity*. In other words, we would like to establish that any protocol, no matter how "intelligent", must always have some minimum amount of communication.

- Of course, the closer the upper and lower bounds are, the better we understand the communication complexity. It's relatively uncommon to get an "exact" answer, and more common to settle on getting the correct scaling (e.g., $\Theta(n)$ or $\Theta(\log n)$).
- For communication complexity specifically, lower bounds are arguably even more important than upper bounds, because most applications of communication complexity in other areas of computer science are in getting lower bounds via reductions.

4.1 Most Functions Require High Communication

As a starting point, we show that *most functions require a high amount of communication*. Specifically, in the case that $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$, the following theorem shows that almost all functions require at

least $n-1$ bits of communication to compute.

Theorem 4.1. *Consider the case that $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ (i.e., both inputs are n-bit and the output is 1-bit). Only a vanishingly small (as $n \to \infty$) fraction of such functions can be computed with $n-2$ bits (or fewer) of communication using deterministic protocols.*

Proof. In accordance with the theorem statement, we consider protocol trees of depth c, where $c \leq n-2$. In general, different leaf nodes may appear at different levels in the tree, but if that's the case, we could always modify the protocol to send additional "dummy" (non-used) bits until exactly c bits have been sent, without modifying the output. Thus, without loss of generality, we may assume that every leaf node lies at depth c (i.e., the protocol always sends exactly c bits).

We proceed to count the number of possible protocol trees. A binary tree with depth c has 2^c leaves, and the number of internal (non-leaf) nodes is $1+2+4+\ldots+2^{c-1} \leq 2^c$. Each internal node is specified by (i) a choice of whether the branch is based on the input of Alice (x) or Bob (y), and (ii) A subset $S \subseteq \{0,1\}^n$ specifying when to send '1' as the bit (for strings outside S, a '0' is sent). For a single internal node, the number of possible choices for (i) and (ii) collectively is at most $2 \cdot 2^{2^n} = 2^{2^n+1} \leq 2^{2^{n+1}}$. Thus, for all internal nodes collectively, the number of choices is at most

$$\left(2^{2^{n+1}}\right)^{2^c} = 2^{2^{n+1} \cdot 2^c} = 2^{2^{n+c+1}}.$$

We also need to consider the leaf nodes. There are 2^c of these, and each one is assigned 0 or 1 (as the output value), meaning there are 2^{2^c} possible assignments in total. Multiplying the number of choices for the internal nodes and the number of choices for the leaf nodes, we conclude that the total number of functions with communication complexity c is at most

$$2^{2^{n+c+1}} \cdot 2^{2^c} = 2^{2^c + 2^{n+c+1}}.$$

Next, we count the number of functions $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$. Each function has 2^{2n} inputs, each of which can produce an output of 0 or 1, so there are $2^{2^{2n}}$ possible functions. Combining with the above calculation, the proportion of functions with communication complexity c is at most

$$\frac{2^{2^c + 2^{n+c+1}}}{2^{2^{2n}}} = 2^{2^c + 2^{n+c+1} - 2^{2n}}.$$

When $c = n-2$, this simplifies to $2^{2^{n-2} + 2^{2n-1} - 2^{2n}}$, and writing $2^{2n-1} = \frac{1}{2} \cdot 2^{2n}$ and $2^{n-2} = o(2^{2n})$, this simplifies to $2^{-\frac{1}{2}2^{2n}(1-o(1))}$. This fraction decays to zero as $n \to \infty$, and in fact does so extremely fast ("doubly exponentially fast"). □

4.2 Lower Bounds from Rectangles

We typically want to understand the communication complexity of specific functions, rather than seeking (overly) general statements as above. Theorem 3.2 gives a useful means for establishing lower bounds in such cases; for convenience, its contrapositive form is repeated as follows.

Corollary 4.2. *If the function f is such that we cannot form a partition of $\mathcal{X} \times \mathcal{Y}$ using at most 2^c monochromatic rectangles, then the communication complexity must exceed c.*

A common strategy is to show that *there are no large monochromatic rectangles*. For example, if $\mathcal{X} = \mathcal{Y} = \{0,1\}^n$ then the size of $\mathcal{X} \times \mathcal{Y}$ is 2^{2n} (i.e., this is the number of entries in the matrix), so if each monochromatic rectangle contains at most γ entries then we require at least $\frac{2^{2n}}{\gamma}$ of them just to "cover" the entire space. We can demonstrate (a slight variation of) this idea through the EQUALS example.

Equality

Consider the equality function EQUALS : $\{0,1\}^n \times \{0,1\}^n \to \{0,1\}$ defined as:

$$\text{EQUALS}(x,y) = \begin{cases} 1 & \text{if } x = y, \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

The trivial protocol has length $n+1$, and we would like to know whether we can do better.

Recall that we represent f as a binary matrix of size $2^n \times 2^n$, and we are interested in its 0-monochromatic and 1-monochromatic rectangles. We observe that EQUALS does in fact have large 0-monochromatic rectangles; see Figure 5 for an example. On the other hand, its 1-monochromatic rectangles are extremely small – just 1×1!

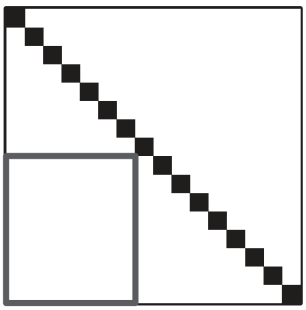


Figure 5: The EQUALS function does have large monochromatic rectangles [1].

Claim 2. *If R is a 1-monochromatic rectangle, then R has size 1×1.*

This claim is fairly straightforward to see – if we try to form a rectangle including (say) two 1s (shaded entries) in the figure shown, it will unavoidably also contain two 0s (unshaded entries).

We know there are 2^n inputs x with $\text{EQUALS}(x,x) = 1$. Thus, we need 2^n rectangles to cover these inputs. In addition, we also need at least one 0-monochromatic rectangle to cover the remaining inputs. By Theorem 3.2, if a function has a communication complexity c, then its domain can be partitioned into 2^c rectangles. Thus, deduce the lower bound $c > n$ and conclude the following.

Theorem 4.3 (Theorem 1.16 in [1]). *The deterministic communication complexity of* EQUALS *is exactly n+1.*

Disjointness

We now turn to an example in which the above approach gives the correct scaling but a suboptimal constant (the optimal constant will be established in Section 4.3). Again consider $x \in \{0,1\}^n$ and $y \in \{0,1\}^n$, and consider the *disjointness* function defined by

$$\text{DISJ}(x,y) = \begin{cases} 1 & \text{if } x_i = y_i = 1 \text{ for some } i \in \{1,\ldots,n\}, \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

In other words, the function is 1 if and only if the sets $\{i : x_i = 1\}$ and $\{i : y_i = 1\}$ are disjoint. Instead of continuing with this form of the function, it will be more convenient to simply define $X = \{i : x_i = 1\}$ and

$Y = \{i : y_i = 1\}$, and proceed with the following *equivalent definition*:

$$\mathrm{DISJ}(X, Y) = \begin{cases} 1 & \text{if } X \cap Y = \varnothing, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

Note that there is a one-to-one mapping between (x, y) and (X, Y). Trivially, Alice can send her whole input to Bob (who can then send back the answer) and achieve a protocol having $n + 1$ bits of communication.

Similar to EQUALS, the function DISJ has "very large" 0-monochromatic rectangles, e.g., $R_i = \{(X, Y) : i \in X, i \in Y\}$ for any fixed i. In fact, it also has "somewhat large" 1-monochromatic rectangles, but only containing 2^n entries which is much smaller than the total matrix size 2^{2n}. Formally:

Claim 3. *Every 1-monochromatic rectangle of* DISJ *contains at most* 2^n *entries.*

Proof. Let $R = A \times B$ be any 1-monochromatic rectangle. Let $X' = \cup_{X \in A} X$ and $Y' = \cup_{Y \in B} Y$ be the union of all the sets of A and B, respectively. By the 1-monochromatic property, we have that X' and Y' are disjoint, and thus $|X'| + |Y'| \leq n$. Since A and B consist of subsets of X' and Y' respectively, we also have $|A| \leq 2^{|X'|}$ and $|B| \leq 2^{|Y'|}$, and we conclude that the number of entries in R is $|A| \cdot |B| \leq 2^{|X'|+|Y'|} \leq 2^n$. □

Next, the number of inputs (X, Y) to DISJ that produce an output of 1 is exactly 3^n. This is because for each element in $\{1, \ldots, n\}$, there are three possibilities: It is only in X, only in Y, or in neither. Overall, at least $3^n/2^n = 2^{(\log_2 3 - 1)n}$ monochromatic rectangles are needed to cover just the 1s of DISJ. Since one further rectangle (at least) is needed for the 0s, we obtain the following.

Theorem 4.4 (**Theorem 1.14 in [1]**). *The deterministic communication complexity of* DISJ *is at least* $\lfloor (\log_2 3 - 1)n \rfloor + 1$.

This lower bound matches the upper bound of $n + 1$ to within a constant factor, but it turns out that we can get a better lower bound via a different approach, which we describe next.

4.3 Fooling Sets

A simple but powerful idea to get improved lower bounds is to *identify a "large" subset of* $S \subset X \times Y$ *such that each element of* S *must be associated with a different rectangle.* This means that the number of rectangles must also be "large", from which Corollary 4.2 gives us a lower bound. The relevant formal definition is given as follows.

Definition 4.1 (**Fooling sets**). *A set* $S \subset X \times Y$ *is called a* ***fooling set*** *for a function* g *if every monochromatic rectangle with respect to* g *can share at most one element with* S.

As hinted above, any partition of the inputs into rectangles must be at least as large as the size of such a fooling set. Thus, Corollary 4.2 immediately implies the following.

Corollary 4.5 (**Theorem 1.20 in [1]**). *If* g *has a fooling set of size* s*, then the communication complexity of* g *is at least* $\log s$.

The natural question is then *how to find a large fooling set.* This is often done case-by-case and typically requires some creativity; we give just one example by returning to the DISJ function.

Disjointness

In the following, we define $N = \{1, \ldots, n\}$ for brevity.

Claim 4. *The set* $S = \{(X, N \setminus X) : X \subseteq [n]\}$ *is a fooling set for* DISJ.

Proof. Recall the definition of a fooling set, and assume for contradiction that there exist two sets $X \neq X'$ such that $(X, N \setminus X)$ and $(X', N \setminus X')$ appear in some rectangle R. Since X and $N \setminus X$ are disjoint, it must be a 1-monochromatic rectangle. By the definition of a rectangle, R must also contain $(X, N \setminus X')$ and $(X', N \setminus X)$. However, this is impossible, since the assumption $X \neq X'$ implies that at least one of $X \setminus X'$ or $X' \setminus X$ is non-empty (and thus DISJ should return 0 instead of 1 on either $(X, N \setminus X')$ or $(X', N \setminus X)$). This establishes the desired contradiction, and we conclude that S must indeed be a fooling set. □

Since $|S| = 2^n$, we conclude from Corollary 4.5 that the deterministic communication complexity is at least n. In fact, we can do slightly better by noting that each $(X, Y) \in S$ has $\text{DISJ}(x, y) = 1$, and we need at least one more 0-monochromatic rectangle on top of the 1-monochromatic ones; this gives the following.

Theorem 4.6 (**Theorem 1.22 in [1]**). *The deterministic communication complexity of* DISJ *is exactly* $n + 1$.

4.4 The Rank Lower Bound Method

The following theorem provides a useful counterpart to the upper bound in Theorem 3.4, albeit with a sizeable gap between the two ($\text{rank} + 1$ vs. $\log_2(\text{rank} + 1)$).

Theorem 4.7 (**Theorem 2.4 in [1]**). *Consider any* $f : \mathcal{X} \times \mathcal{Y} \to \{0, 1\}$*, and let* M *be the associated binary matrix of size* $|\mathcal{X}| \times |\mathcal{Y}|$*. If* M *is not the all* 1*s matrix, then the communication complexity of* f *is at least* $\log_2(\text{rank}(M) + 1)$.

Proof. Let c denote the communication complexity. Theorem 3.2 shows that M can be partitioned into at most 2^c monochromatic rectangles. For each rectangle R in the partition, let z_R denote the value of the function in the rectangle, and let M_R denote the $|\mathcal{X}| \times |\mathcal{Y}|$ matrix whose (x, y)-th entry is z_R if $(x, y) \in R$, and 0 otherwise. Observe that:

- If $z_R = 0$, then M_R contains all 0s and thus has rank 0;
- If $z_R = 1$, then M_R contains a single rectangle of 1s, and thus has rank 1 (see item #2 in Fact 2 and note that every non-zero column is identical).

In both cases, we have $\text{rank}(M_R) \leq 1$. Moreover, M can be expressed as the sum of 2^c such matrices, and at least one of these matrices is 0 because M has at least one zero. Thus, by item #4 in Fact 2, we have $\text{rank}(M) \leq 2^c - 1$, and thus $c \geq \log_2(\text{rank}(M) + 1)$. □

Note that the condition that M is not all-1s is necessary, since the all-1s case gives $\log(\text{rank}(M) + 1) = 1$ but its communication complexity is trivially 0.

As an example application of Theorem 4.7, we return to the EQUALS function and give an alternative proof of its lower bound: The matrix M is simply the $2^n \times 2^n$ identity matrix, which has rank 2^n. Thus, the rank-based lower bound becomes $\lceil \log(2^n + 1) \rceil = n + 1$.

(Optional**)** Recent research has shown that the rank-based upper bound can be improved to at least to at least $\widetilde{O}(\sqrt{r})$. It remains an open problem as to whether the upper bound can be improved to

$(\log r)^{O(1)}$ – this is called the *log-rank conjecture*. If the answer is "yes", then it is known that the $O(1)$ term must be at least 2, since there exist cases where the communication complexity is $\widetilde{\Omega}((\log r)^2)$. See the end of https://www.youtube.com/watch?v=zFHWmmThdT4 for further discussion.

5 Randomized Communication Complexity

Randomized algorithms are central to theoretical computer science, and also widely used in practice. Accordingly, it is natural to ask to what extent *randomized protocols* can help when it comes to communicating $f(x, y)$. It turns out that this can in fact change the picture quite drastically. While this is a major topic in communication complexity, we give it only a brief treatment, omitting most proofs. We first introduce some important concepts.

Randomization and error probability. By randomization, we mean that Alice and Bob are now allowed to make decisions such as "Send a 1 with probability p, and a 0 with probability $1-p$". (The source of randomness is important, and is discussed below.) Naturally, this means that multiple invocations of the protocol for fixed (x, y) might produce different outputs. We fix $\delta > 0$ and impose that for all inputs, the protocol is correct with probability at least $1 - \delta$:

$$\max_{x \in \mathcal{X}, y \in \mathcal{Y}} \mathbb{P}[\pi(x, y) \neq f(x, y)] \leq \delta, \tag{4}$$

where $\pi(x, y)$ is the (random) output of the protocol. Typically we consider δ to be a fixed constant in $\left(0, \frac{1}{2}\right)$, such as $1/3$, since a success probability of $2/3$ (say) can easily be boosted to any target $1 - \delta$ by repeating it $O\left(\log \frac{1}{\delta}\right)$ times and taking the majority output. (**Exercise:** *Try proving this using results from the concentration lecture.*)

Number of bits communicated. The notion of "number of bits communicated" can also be defined in one of several ways, since it is now a random variable. Specifically, we could consider its average, its high-probability behavior, or its worst-case behavior. It turns out that this choice is fairly inconsequential, at least in terms of scaling laws (**Exercise:** *Think about why this is the case*), and in the following we will take the requirement "at most c bits are communicated" as needing to hold *with probability one*.

Source of randomness: There are at least 3 distinct models for the source of randomness:

- **Shared randomness (the public-coin model)**: A protocol is said to use public coins if all parties have access to a common sequence of random bits.
- **Private randomness (the private-coin model)**: A protocol is said to use private coins if each party privately and independently samples a string of random bits.
- **Input randomness (the distributional model)**: Another approach is to assume that the *input* is random, and characterize how many bits need to be communicated *on average*. (Note: In this case, the above considerations, including (4), need to be suitably modified.)

Usually, when people refer to "the randomized communication complexity", they are referring to the public-coin case. This is partially justified by the following result, which shows that converting public-coin to private-coin loses at most a logarithmic factor (which is typically considered "small").

Theorem 5.1. (Newman's Theorem) *Any pubic-coin protocol can be converted to a private-coin protocol with at most a δ' increase in the error probability and at most an $O\left(\log n + \log \frac{1}{\delta'}\right)$ increase in the communication cost (for any $\delta' > 0$).*

In particular, for a constant error probability, if the public-coin communication complexity is $\Theta(\log n)$, then its private-coin communication complexity is $\Theta(\log n)$, and if its public-coin communicaton complexity is $\omega(\log n)$ (e.g., $\Theta(\sqrt{n})$), then the private-coin communication complexity is identical up to asymptotically lower-order terms. Of course, any private-coin protocol can trivially be converted to a public-coin protocol.

Equality

While EQUALS served as a "very hard" function (requiring $n+1$ bits of communciation) for determinstic protocols, it turns out that this property is lost in the randomized case. This is seen via the following upper bounds for the public-coin and private-coin cases respectively.

Theorem 5.2. *The public-coin randomized communication complexity of EQUALS is $O(1)$.*

Theorem 5.3. *The private-coin randomized communication complexity of EQUALS is $O(\log n)$.*

Public-coin case: We first consider the public coin case, and consider the following protocol:

- Alice and Bob extract $2n$ random bits, say $r = (r_1, \ldots, r_n)$ and $r' = (r'_1, \ldots, r'_n)$ (with $r_i \sim$ Bernoulli(1/2) independently and similarly for r') from the public source of randomness. Thus, the same random strings r, r' are known to both of them.
- Alice computes the inner products $r \cdot x$ and $r' \cdot x$ and sends them to Bob.
- Bob computes $r \cdot y$ and $r' \cdot y$. If $r \cdot x = r \cdot y$ and $r' \cdot x = r' \cdot y$ then Bob declares that $x = y$; otherwise he declares that $x \neq y$.

If $x = y$, then the correct decision will trivially be made with probability one. On the other hand, if $x \neq y$, it is easy to check that $\mathbb{P}[r \cdot x = r \cdot y] = \frac{1}{2}$ and similarly for r', and thus a correct decision is made with probability $1 - \left(\frac{1}{2}\right)^2 = \frac{3}{4}$. Observe that this probability can be boosted to $1 - 2^{-k}$ by using k length-n strings instead of just two.

Private-coin case: We could infer the private-coin result from the public-coin one using Theorem 5.1, but it's also interesting to give a direct proof. The above strategy cannot directly be used, since Alice's random bit string r is not known to Bob, and sending it would be too expensive.

Instead, we consider the following strategy:

- Alice and Bob agree (in advance) on using a binary error-correcting code for length-n messages, producing length-βn codewords for some $\beta > 1$, and satisfying the property that any two codewords $C(x)$ and $C(y)$ (for $x \neq y$) differ in at least a fraction α of their bits. (We know from the earlier lecture that such codes exist for suitable constants α and β.)
- Alice computes $C(x)$, and sends the pair $(i, C(x)_i)$ for k randomly-chosen indices $i \in \{1, \ldots, \beta n\}$.
- Bob computes $C(y)$, and checks if $(i, C(x)_i) = (i, C(y)_i)$ for all such i values. If so, he declares $x = y$, and otherwise he declares $x \neq y$.

We again have a probability one of being correct in the $x = y$ case, and it is a simple exercise to show that we succeed with probability at least 3/4 in the $x \neq y$ case as long as k is large enough as a function of (α, β). (**Exercise:** *Try showing this.*) Sending an index in $\{1, \ldots, \beta n\}$ requires $\lceil \log_2 \beta n \rceil$ bits (plus 1 bit for $C(x)_i$), so the communication cost is $O(\log n)$ when $k = O(1)$.

Disjointness

With EQUALS being an "easy" problem in the randomized case, we need to search for different problems that are "hard" (as is required for establishing lower bounds via reductions). The most prominent such problem in the randomized case is **disjointness**, somewhat like how SAT is the starting point for NP-hardness reductions. *(Side-note: Another prominent example is "inner product mod 2", but we'll skip that.)*

The hardness result is formally stated as follows.

Theorem 5.4 (**Theorem 6.19 in [1]**). *Any randomized (public-coin) protocol that computes the disjointness function with error probability at most* $1/2 - \epsilon$ *must have communication complexity* $\Omega(n\epsilon^2)$ *(in particular,* $\Omega(n)$ *if* ϵ *is a constant).*

This matches the trivial upper bound of $O(n)$ (via Alice sending her whole input) to within a constant factor. The proof of this theorem is non-trivial and relies heavily on tools from information theory, but interested students should now have the prerequisite background to be able to follow it; see [1, Chapter 6] for a self-contained proof.

Discussion

We have only scratched the surface of randomized communication complexity. Randomized protocols for specific functions often require some creativity (e.g., see the example 'Sparse Set Disjointness' in `https://cseweb.ucsd.edu/classes/wi19/cse291-b/3-randomized.pdf`, or the function "$\geq$" in the tutorial). There are also a number of other variations of communication complexity, some of which we briefly mention in Appendix A.

6 Applications in Theoretical Computer Science

Communication complexity has proven to be an powerful tool for proving lower bounds in other areas of theoretical computer science (TCS), including VLSI, decision tree lower bounds, cell probe model and dynamic data structures, Boolean circuit complexity, Turing machine time-space trade-offs, streaming algorithms, game theory, high-dimensional estimation, distributed computation, and proof complexity. (Don't worry if you haven't heard of most of these.)

In the following, we give just two examples of how lower bounds on communication complexity imply certain lower bounds in computer science problems of interest.

6.1 Example 1: Space Complexity Lower Bounds for Streaming Algorithms

- Loosely speaking, a streaming algorithm is one that reads in a list sequentially and outputs some property of that list, but is unable to store the entire list in memory. Thus, it has to "discard" some elements along the way.
- As a simple example, if the goal is to output the average of a list of numbers, then only two things need to be known – the sum of all elements, and the number of elements (then output the ratio of the two). When a new element of list is read, these numbers can be updated and then the list element can be forgotten.

- For other problems, even simple ones, it can be very unclear how much space is required (e.g., if the goal is median instead of mean, this is already much less obvious).
- Reductions based on communication complexity are a powerful method to prove lower bounds, showing a certain minimum amount of space is unavoidable.

Deterministic algorithm example: Number of distinct elements

- Consider the following goal: Given a list of numbers $a_1, \ldots, a_n$, each taking values in the range $\{1, 2, \ldots, m\}$, output the number of distinct values appearing in the list. For example, if the list is $(1, 3, 1, 5, 9, 5, 5, 3, 5)$, then there are 4 distinct elements: 1,3,5,9.
- **Claim:** If $m \geq n$, then any deterministic streaming algorithm for this task must use $\Omega(n)$ space.
  - The more general result is $\Theta(\min\{m, n\})$, and it is easy to get a near-matching upper bound (just store the entire list if $\min\{m, n\} = n$, or store an "indicator bit" for each value if $\min\{m, n\} = m$).
- To see this, we will perform a reduction based on the $\Omega(n)$ communication complexity of the EQUALS function. Fix any inputs (x, y) to the EQUALS problem, with length n each. Then consider the following two lists of numbers:

$$L_x = \{i : x_i = 1\}$$
$$L_y = \{i : y_i = 1\}.$$

  These lists both have size at most n.
- Consider the following protocol for EQUALS:
  - Alice runs streaming Distinct Elements on L_x, and Bob runs it on L_y (this requires no communication).
  - Alice then sends the memory contents of the Distinct Elements algorithm (after L_x has been processed) to Bob, who continues running it (with further elements from L_y) to get the number of Distinct Elements in $L_x \circ L_y$ (where $\circ$ denotes list concatenation).
  - Alice also sends the number of Distinct Elements in L_x to Bob. (This only requires $O(\log n)$ bits of communication.)
  - Bob checks whether L_x, L_y, and $L_x \cup L_y$ have the same number of distinct elements. If so, then he declares that EQUALS(x, y)=1, otherwise 0.

  It is easy to see that this produces the correct output of EQUALS. Yet if the streaming algorithm's memory were $o(n)$, then the same would be true of the communication cost.
- We know that $o(n)$ communication cost is impossible, so we conclude that the streaming algorithm must have $\Omega(n)$ memory.

Randomized algorithm example: Approximate highest count

- Consider the following goal: We are given a list of numbers $a_1, \ldots, a_n$, each taking values in the range $\{1, 2, \ldots, m\}$. Let n_i be the number of occurrences of symbol i for $i = 1, \ldots, m$, and define $n_{\max} = \max_i n_i$ to be the highest count. The goal is to output a number $\hat{n}$ such that $\hat{n} \in [0.8n_{\max}, 1.2n_{\max}]$ (i.e., find an approximate value of $n_{\max}$) with probability at least 2/3.

- The exact numbers 0.8, 1.2 and 2/3 aren't important, but they provide leniency compared to the above example – we only need to give an approximate answer, and we can use a randomized algorithm that has some probability of failing.

- **Claim:** If $m \geq n$, then any (possibly randomized) streaming algorithm achieving this goal must use $\Omega(n)$ space. (Again, the more general dependence is $\min\{m, n\}$.)

- The proof is quite similar to before, but is based on the hardness of DISJOINT rather than EQUALS (the latter is no longer suitable, since we are now allowing randomization, and EQUALS has very low randomized communication complexity).

- Specifically, we use the streaming algorithm $\mathcal{A}$ as a subroutine for computing DISJOINT(x, y):
  - Alice feeds into $\mathcal{A}$ the indices i where $x_i = 1$;
  - Alice sends the memory contents of $\mathcal{A}$ to Bob;
  - Starting with those memory contents, Bob feeds into $\mathcal{A}$ the indices i where $y_i = 1$, then reads the algorithm's output;
  - Bob declares DISJOINT(x, y)=1 if the output is at most 1.5, and 0 otherwise.

- This gives the correct answer for DISJOINT with probability at least 2/3, because $n_{\max}$ is 1 in the disjoint case (meaning the answer should be in $[0.8, 1.2]$) but is at least 2 in the non-disjoint case (meaning the answer should be 1.6 or higher).

- We know that DISJOINT has communication complexity $\Omega(n)$, so we conclude that the communication step (consisting of the memory contents) must have $\Omega(n)$ bits. That is, $\mathcal{A}$ requires $\Omega(n)$ memory.

6.2 (**Optional**) Example 2: Query Complexity of Property Testing

- *Property testing* is a broad field in computer science and statistics where we seek to determine whether a given string/function/random variable/etc. has a certain property using limited information (e.g., queries to the string/function, or samples from the distribution).

- Here we focus on the problem of *motonicity testing*:
  - There exists a function $f(x)$ with inputs in $\{0, 1\}^n$ and outputs in $\{0, 1, 2, \ldots, R\}$ for some $R > 0$.
  - The function f is not known directly, but instead the algorithm can can only perform *queries*: If input x is queried, then the value $f(x)$ is returned. We would like to use as few queries as possible. (Ideally far fewer than the trivial 2^n attained by querying every point.)
  - The function f is said to be *monotone* if changing any coordinate of the input from 0 to 1 increases the function value (or keeps it the same).
  - **Goal:** Construct a (possibly randomized) algorithm that performs Q queries and then returns:
    - 'YES' with probability at least 2/3 if f is monotone;
    - 'NO' with probability at least 2/3 is f is ϵ-far from being monotone (i.e., we would need to modify at least an ϵ-fraction of function values to get a monotone function);
    - (Either 'YES' or 'NO' may be returned in all other cases.)

- **Claim:** When $R = 2(n+1)$ (or larger) and $\epsilon = \frac{1}{8}$ (or smaller), this problem requires $Q = \Omega(n)$ queries.
  - This nearly matches a known upper bound of $O\big(\frac{n}{\epsilon} \log R\big)$, e.g., see Chapter 8 of Roughgarden.
- The proof is again based on a reduction involving (a variant of) the DISJOINT problem:
  - It will be convenient to treat binary vectors and sets interchangeably, so we can write $f(x)$ equivalently as $f(S)$, with S being the set of indices where $x_i = 1$.
  - Define $V = \{1, \ldots, n\}$, let A and B be subsets of V, and consider the following function defined on subsets of $\{0,1\}^n$:
    $$h_{AB}(S) = 2|S| + (-1)^{|S \cap A|} + (-1)^{|S \cap B|}.$$
    - Observe that if A and B are disjoint, then h_{AB} is monotone, because adding an element to S adds 2 in the first term, but only one of the other two terms can decrease from $+1$ to -1 (not both).
    - If A and B have exactly one element in common, then it can be shown that h_{AB} is $\frac{1}{8}$-far from monotone; roughly, this is because if $|A \cap B| = \{i\}$ for some i, then $\frac{1}{4}$ of the possible subsets of $\{1, \ldots, n\} \setminus \{i\}$ will give $h_{AB}(S \cup \{i\}) < h_{AB}(S)$ due to both $|S \cap A|$ and $|S \cap A|$ being even (and hence changing to odd when i is added).
    - Observe also that $h_{AB}(S)$ outputs an integer between 0 and $2n + 2$, which is consistent with the choice $R = 2(n + 1)$ above.
  - The reduction actually uses a variant of DISJOINT called UniqueDisjoint, where in the case of being non-disjoint, the strings (x, y) are guaranteed to only differ in one entry. The $\Omega(n)$ communication complexity lower bound is known to still hold for this problem.
  - The reduction is as follows:
    - Alice knows $x \in \{0,1\}^n$ and the corresponding set is called A. Bob knows $y \in \{0,1\}^n$ and the corresponding set is called B.
    - Alice and Bob both run monotonicity testing on h_{AB}; after each query S, Alice sends $(-1)^{|S \cap A|}$ to Bob, and Bob sends $(-1)^{|S \cap B|}$ to Alice, so that both of them get the query response $h_{AB}(S)$.
    - If the monotonicity tester returns 'YES' then A and B are declared to be disjoint. Otherwise they are declared to overlap in a single entry.
  - Observe that the total amount of communication is $2Q$ when Q queries are made.
  - The fact that UniqueDisjoint has (randomized) communication complexity $\Omega(n)$ thus directly implies that we need $Q = \Omega(n)$.

A (**Optional**) Other Communication Models and Settings

Beyond the settings we have covered, there are a number of other settings of interest, some of which we mention as follows:

- **Average-case complexity:** While the most common settings consider the worst-case communication cost over all x and y, there are also cases where an average-cost notion is of more interest. Specifically, we can assume that $(x, y) \sim D$ for some joint distribution D, and then study the *average* number of bits exchanged with respect to D.

- **Non-Boolean functions:** Many functions that we are interested in computing have non-binary outputs, say represented by an m-bit string for some $m > 1$. For example, the Hamming distance between $x \in \{0,1\}^n$ and $y \in \{0,1\}^n$ is represented by a number in $\{0, 1, \ldots, n\}$, which can be represented using $\lceil \log_2(n+1) \rceil$ bits. See for example [7].

- **One-way communication:** In the one-way setting, communication is only allowed from Alice to Bob, not vice versa. The goal is for Alice to communicate as little as possible while ensuring that Bob can compute $f(x, y)$. This setup moves things somewhat closer to information theory, and its associated tools like Fano's inequality are common.

- **Multi-agent settings:** Naturally, we can consider more than two agents, e.g., there could be k agents each knowing one of $x_1, \ldots, x_k$ (say with each $x_i \in \{0,1\}^n$), and the goal is to compute $f(x_1, \ldots, x_k)$. For the communication model, each agent could be allowed to broadcoast bits to all other agents, or communication could be more constrained (e.g., only allowed to occur across certain edges in a graph).

- **Nondeterministic protocols:** *(Note: The word "nondeterministic" here should be interpreted similarly to the complexity class NP; it is not synonymous with "randomized".)*

  In a nondeterministic protocol, the players are both provided an additional third input z (interpreted as a "nondeterministic guess"). Apart from this guess, the protocol is deterministic. The cost incurred on x, y is

  $$\min_z \{|z| + \text{number of bits exchanged by the protocol when the guess is } z\}.$$

  The *nondeterministic communication complexity* of f is the minimum k such that there is a nondeterministic protocol whose cost for all input pairs is at most k.

B (**Optional**) Proof of Theorem 3.3 (Rectangles to Protocols)

Recall that both Alice and Bob know f. We can similarly assume that they both know the partitioning into monochromatic rectangles, which we denote by $\mathcal{R}$. The goal is to devise a protocol that identifies the unique rectangle in $\mathcal{R}$ containing (x, y), since this clearly amounts to knowing the function value.

Let the rectangles be arbitrarily indexed as $1, 2, \ldots, 2^c$ (or fewer). We consider a protocol that operates in rounds, where in each round either Alice or Bob sends the index of one such rectangle (we will later convert from "sending indices" to "sending bits"). Specifically, this will be done in a manner such that the agents can eliminate at least half of the remaining rectangles (i.e., conclude that they do *not* contain (x, y)). This will mean that after at most $\log_2(2^c) = c$ rounds, we have narrowed down to the single (correct) rectangle containing (x, y), impying that $f(x, y)$ is known.

To achieve the above goal, we introduce the notion of *good rectangles*. Before doing so, we define the following notions (illustrated in Figure 6):

- Two rectangles $R = A \times B$ and $R' = A' \times B'$ *intersect horizontally* if A intersects A';
- Two rectangles $R = A \times B$ and $R' = A' \times B'$ *intersect vertically* if B intersects B'.

An important observation is that if x belongs to both A and A' (i.e., $x \in A \cap A'$) and y belongs to both B and B' (i.e., $y \in B \cap B'$), then (x, y) will lie in both $A \times B$ and $A' \times B'$. This implies the following:

Fact 3. *Two **disjoint** rectangles cannot intersect both horizontally and vertically.*

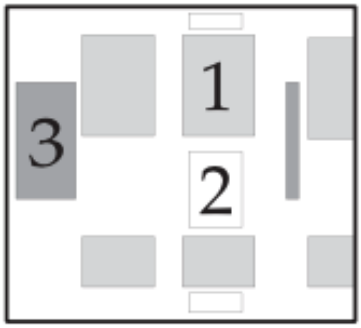


Figure 6: Rectangles 1 and 2 intersect vertically, while rectangles 1 and 3 intersect horizontally. [1]

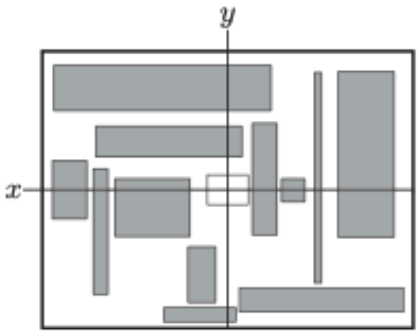


Figure 7: Either a rectangle aligned with x intersects no more than half of the other rectangles horizontally, or a rectangle aligned with y intersects at most half of them vertically, as detailed in [1].

Definition B.1 (**Good Rectangles** [1]). *For a given input (x, y), a rectangle $R = (A \times B) \in \mathcal{R}$ is **horizontally good** if $x \in A$ and R horizontally intersects at most $|\mathcal{R}|/2$ rectangles in $\mathcal{R}$. Similarly, R is **vertically good** if $y \in B$ and R vertically intersects at most $|\mathcal{R}|/2$ rectangles in $\mathcal{R}$. We say that R is **good** if it is either horizontally good or vertically good.*

Observe that if Alice can identify a horizontally good rectangle R, announcing its *name* will reduce the set of considered rectangles by at least half (since the ones that *don't* horizontally intersect R can be removed). Likewise, Bob can achieve a similar reduction by announcing a vertically good rectangle. Given that we assume all parties possess infinite computational capabilities, we don't dwell on the process by which they identify these good rectangles.

The remaining question is: *Is it always possible to find a good rectangle?* The answer is YES, as the following claim establishes.

Claim 5. *The unique rectangle in $\mathcal{R}$ that contains (x, y), denoted as $R_{x,y}$, is good.*

Proof. From Fact 3, we know that no rectangle in $\mathcal{R}$ can intersect $R_{x,y}$ both horizontally and vertically. Consequently, either at most half of the rectangles in $\mathcal{R}$ intersect $R_{x,y}$ horizontally (thus making it *horizontally good*), or at most half intersect it vertically (thus making it *vertically good*). This establishes the claim, and Figure 7 gives an illustration. □

A slight subtlety is that both Alice and Bob might know a good rectangle, and the protocol needs to specify who should send first. A simple approach is to let them take turns in giving them the opportunity to send a rectangle's index; then:

- If they know a good rectangle, they send '1' followed by the rectangle's index;
- If they don't know a good rectangle, they send '0'.

In this manner, there will be at most 2 extra bits sent for each rectangle index sent. The final communication cost is computed by noting that (i) with at most 2^c rectangles, sending a unique index requires at most $\log_2(2^c) = c$ bits; and (ii) with 2^c candidates initially and halving (or better) each round, there are at most $\log_2(2^c) = c$ rounds. Thus, the total communication cost is $O(c^2)$.

References

[1] A. Rao and A. Yehudayoff, *Communication Complexity and Applications*. Cambridge University Press, 2020.

[2] A. A. Razborov, "Communication complexity," in *An Invitation to Mathematics: From Competitions to Research*. Springer, 2011, pp. 97–117.

[3] T. Roughgarden *et al.*, *Communication complexity (for algorithm designers)*. Now Publishers, Inc., 2016, vol. 11, no. 3–4.

[4] T. Lee, A. Shraibman *et al.*, "Lower bounds in communication complexity," *Foundations and Trends® in Theoretical Computer Science*, vol. 3, no. 4, pp. 263–399, 2009.

[5] A. C.-C. Yao, "Probabilistic computations: Toward a unified measure of complexity," in *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*. IEEE Computer Society, 1977, pp. 222–227.

[6] R. Kothari, "Nearly optimal separations between communication (or query) complexity and partitions," *arXiv preprint arXiv:1512.01210*, 2015.

[7] L. Fontes, S. Laplante, M. Lauriere, and A. Nolin, "The communication complexity of functions with large outputs," in *International Colloquium on Structural Information and Communication Complexity*. Springer, 2023, pp. 427–458.

CS5275 Lecture 12: Other Topics

Jonathan Scarlett

March 24, 2025

Useful references:

- CMU course: https://www.youtube.com/playlist?list=PLm3J0oaFux3ZYpFLwwrlv_EHH9wtH6pnX
- USyd lectures: https://ccanonne.github.io/teaching/COMPx270
- Moore, C. and Mertens, S., 2011. *The nature of computation.* OUP Oxford.
- Wigderson, A., 2019. *Mathematics and computation: A theory revolutionizing technology and science.* Princeton University Press.

This final set of notes outlines some topics that we might have delved into if we had more time. The first two sections are in the category of mathematical basics/fundamentals, whereas the subsequent sections are on more specialized topics, most of which could easily fill an entire course.

Note on examination: None of the material in this lecture is examinable.

1 Norms and Distances

1.1 Vector Norms

The vector norm that we use most (and coincides with our real-world understanding of "length") is the ℓ_2 norm $\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^n x_i^2}$, also known as Euclidean norm. Beyond ℓ_2, the most common vector norms are:

- ℓ_1 norm: $\|\mathbf{x}\|_2 = \sum_{i=1}^n |x_i|$. This is less sensitive to outliers than the ℓ_2 norm, and it has also served as a very useful "convex proxy" for the non-convex notion of sparsity (number of non-zeros).
- ℓ_∞ norm: $\|\mathbf{x}\|_\infty = \max_{i=1,\dots,n} |x_i|$
- More generally, for $p \in [1,\infty)$ the ℓ_p norm is $\|\mathbf{x}\|_p = \big(\sum_{i=1}^n |x_i|^p\big)^{1/p}$ (taking the limit $p \to \infty$ can be shown to give $\|\mathbf{x}\|_\infty$).

It is useful to be aware of inequalities between these norms, e.g.:

$$\|\mathbf{x}\|_2 \le \|\mathbf{x}\|_1 \le \sqrt{n}\|\mathbf{x}\|_2$$
$$\|\mathbf{x}\|_\infty \le \|\mathbf{x}\|_2 \le \sqrt{n}\|\mathbf{x}\|_\infty,$$

where n is the length of $\mathbf{x}$. In addition, *Hölder's inequality* states that $|\langle \mathbf{u}, \mathbf{v}\rangle| \le \|\mathbf{u}\|_p \|\mathbf{v}\|_q$ when $p, q \ge 1$ satisfy $\frac{1}{p} + \frac{1}{q} = 1$; setting $p = q = 2$ gives Cauchy-Schwarz.

1.2 Matrix Norms

Some common matrix norms are listed as follows (letting $\mathbf{A}$ be a generic matrix), without going into detail. We will mention the notion of *singular values*, which are briefly defined in the next section.

To distinguish between matrix norms and vector norms, here we use $|||\cdot|||$ for matrices (but it's much more common to just re-use $\|\cdot\|$):

- *Spectral norm* $|||\mathbf{A}|||_2$: This is the largest singular value, and can also be written as $|||\mathbf{A}|||_2 = \sup_{\mathbf{x}\neq\mathbf{0}} \frac{\|\mathbf{A}\mathbf{x}\|_2}{\|\mathbf{x}\|_2}$, which implies the useful property $\|\mathbf{A}\mathbf{x}\|_2 \leq |||\mathbf{A}|||_2 \|\mathbf{x}\|_2$.
- *Operator norm*: Generalizing the above equation for $|||\mathbf{A}|||_2$, we can consider $|||\mathbf{A}|||_{p\to q} = \sup_{\mathbf{x}\neq\mathbf{0}} \frac{\|\mathbf{A}\mathbf{x}\|_q}{\|\mathbf{x}\|_p}$.
- *Nuclear (trace) norm*: This is the sum of singular values. It is a useful convex proxy for the rank of a matrix, analogous to the ℓ_1 norm for vectors.
- *Frobenius norm*: $|||\mathbf{A}|||_F = \sqrt{\sum_{i,j} |A_{ij}|^2}$. In other words, re-arrange $\mathbf{A}$ into a length-(mn) vector and take its Euclidean norm.

1.3 Distances and Divergences Between Probability Measures

There are many useful ways to measure how different two probability distributions are. Some of the most common ones are as follows:

- Total variation distance, which is $\frac{1}{2}\sum_x |P(x) - Q(x)|$ for PMFs, $\frac{1}{2}\int |P(x) - Q(x)|dx$ for PDFs, and $\sup_A |\mathbb{P}_P[A] - \mathbb{P}_Q[A]|$ (where A is an arbitrary event) in general.
- KL divergence: $D_{\mathrm{KL}}(P\|Q) = \mathbb{E}_{X\sim P}\big[\log \frac{P(X)}{Q(X)}\big]$.
- Hellinger distance and χ^2 divergence (definitions omitted)
- Wasserstein distances / earth-mover distances, which can roughly be understood as follows: If we interpret two probability density (or mass) functions as suitably-shaped "piles of dirt", what's the smallest possible total distance of dirt moved to transform pile P into pile Q?

Different measures have different desirable properties, which is why we often need to work with multiple, or use inequalities (e.g., Pinkser's inequality: $d_{\mathrm{TV}} \leq \sqrt{D_{\mathrm{KL}}/2}$) to "convert" from one to another. Desirable inequalities include triangle inequality (e.g., for TV), tensorization inequalities that relate $d(\prod_i P_i, \prod_i Q_i)$ to individual $d(P_i, Q_i)$ (e.g., for KL), and data processing inequalities (stating that $d(P\circ T, Q\circ T) \leq d(P, Q)$ for any transformation T).

Some example uses of the above measures are as follows:

- Total variation is useful for moving from a hard-to-analyze distribution to an easier one via $\mathbb{P}_P[A] \leq \mathbb{P}_Q[A] + \|P - Q\|_{\mathrm{TV}}$. For example, P might be a multinomial distribution, and Q might be its (simpler) Poisson approximation.
- For KL divergence, if we sample n i.i.d. random variables from a PMF Q and ask what the probability is that the proportions of symbols instead match P, the answer turns out to be roughly $e^{-nD(P\|Q)}$. See *Sanov's Theorem* for a more general statement.
- Hellinger and χ^2 measures arise in hypothesis testing and proving lower bounds.

- Wasserstein distance is useful when the *closeness of values* matters and not only the closeness of probabilities. For example, if P and Q are PMFs only taking values in $\{a, b\}$ (e.g., $a = 0$ and $b = 1$), then all choices of $(a, b) \in \mathbb{R}^2$ (with $a \neq b$) will give identical TV distance, KL divergence, etc., whereas the Wasserstein distance will depend on whether a and b themselves are close or far.

2 Matrix Decompositions

- If there's one matrix decomposition that everyone should be familiar with, it is *Singular Value Decomposition*: An $m \times n$ complex-valued matrix $\mathbf{A}$ can be decomposed as follows (with $(\cdot)^*$ denoting conjugate transpose):

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^* = \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^*$$

  for some unitary matrices $\mathbf{U}$ (size $m \times m$) and $\mathbf{V}$ (size $n \times n$) and a "rectangular diagonal" matrix $\mathbf{\Sigma}$ (size $m \times n$), i.e., all entries off the diagonal are zero. Those diagonals $\{\sigma_i\}$ are called *singular values*, and $r \leq \min\{m, n\}$ is the number of non-zero singular values and the rank of $\mathbf{A}$.

  - Interpretation: Any linear transform ($\mathbf{A}$) can be expressed as a rotation ($\mathbf{V}^*$) followed by scaling ($\mathbf{\Sigma}$) followed by another rotation ($\mathbf{U}$).
  - When $\mathbf{A}$ is a real symmetric matrix, this simplifies to the *eigenvalue decomposition*: $\mathbf{A} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^{-1}$, where $\mathbf{Q}$ is an orthogonal matrix containng eigenvectors, and $\mathbf{\Lambda}$ is a diagonal matrix containing eigenvalues of $\mathbf{A}$.
  - More generally, the singular values of $\mathbf{A}$ are the square roots of eigenvalues of $\mathbf{A}^*\mathbf{A}$.

- An example use of SVD is to approximate a large matrix by a more compact *low-rank form* by replacing "small" singular values by 0. For example, this is the idea of *Principal Component Analysis*; if k non-zero singular values are kept, we can interpret this as projecting the data onto the space spanned by the k "most significant directions", and ignoring the other "less significant" directions.

- Other useful decompositions include QR, LU, and Cholesky (e.g., a summary can be found on the Wiki page `https://en.wikipedia.org/wiki/Matrix_decomposition`)

- A useful resource for matrices in general is the Matrix Cookbook: (`https://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf`)

- **(Relevant course: MA4230 Matrix Computation)**

3 Further Probabilistic Limit Theorems

The most well-known probabilistic limit theorems are law of large numbers, central limit theorem, and concentration bounds (e.g., Hoeffding or Chernoff bounds). In addition to these, there are a number of lesser-known bounds that are worth being aware of:

- **Berry-Esseen theorem:** This is a non-asymptotic version of the central limit theorem (CLT), which can be useful since avoiding asymptotics can keep analysis neater and avoid confusions with things like the order of limits.

An example statement is as follows: If $X_1, \ldots, X_n$ are i.i.d. with mean zero, variance σ^2, and $\mathbb{E}[|X|^3] = \rho < \infty$, then letting $Y_n = \frac{1}{n}\sum_{i=1}^n X_i$ and letting F_n be the CDF of $\frac{Y_n\sqrt{n}}{\sigma}$, we have the following non-asymptotic normality result:

$$\sup_x \big|F_n(x) - \Phi(x)\big| \le \frac{C\rho}{\sigma^3\sqrt{n}},$$

where C is a (not too large) constant and Φ is the CDF of $N(0,1)$. A similar statement also holds for probabilities $\sup_A \big|P_n[A] - P_{\mathrm{N}(0,1)}[A]\big|$ (with A being events) rather than CDFs.

- **Uniform convergence of the CDF.** Let $X_1, \ldots, X_n$ be an i.i.d. sequence with each X_i having CDF F_X. By the law of large numbers, for any threshold x, the proportion of X_i's satisfying $X_i \le x$ will become close to $F_X(x)$ for large n (with high probability). If we are interested in this holding for *multiple* x values, the simplest approach would be to try a union bound. But then we are limited in how many values we can consider, and this approach may preclude using a threshold that itself depends on the i.i.d. sequence.

  These limitations are overcome by results on *uniform convergence of the CDF*; in particular:

  - The *Glivenko-Cantelli theorem* states that with probability one, it holds that $\sup_{x\in\mathbb{R}} |F_n(x) - F_X(x)| \to 0$ as $n \to \infty$, where F_n is the empirical CDF.
  - The *Dvoretzky-Kiefer-Wolfowitz (DKW) theorem* gives a non-asymptotic version, $\mathbb{P}\big[\sup_x |F_n(x) - F_X(x)| \ge \epsilon\big] \le Ce^{-2n\epsilon^2}$ (initially proved with unspecified C, and later with $C = 2$).

  There are also extensions of these results to multivariate CDFs.

- **Moderate deviations:** Roughly speaking, the CLT concerns $\frac{1}{\sqrt{n}}$ deviations from the mean and constant probabilities, while large deviations results concern constant deviations from the mean and exponentially small probabilities. Moderate deviations results concern regimes in-between – ϵ_n deviations with $\frac{1}{\sqrt{n}} \ll \epsilon_n \ll 1$, and probabilities that decay but slower than exponentially. Under suitable assumptions on the random variable being not too heavy-tailed, we have for i.i.d. X_i that

$$\mathbb{P}\bigg[\bigg|\frac{1}{n}\sum_{i=1}^n \big(X_i - \mu\big)\bigg| \ge \epsilon_n\bigg] \le \exp\Big(-\Theta(n\epsilon_n^2)\Big).$$

  Perhaps the most common choice is to set $\epsilon_n = \sqrt{\frac{C\log n}{n}}$ for some constant $C > 0$, in which case we get $\Theta(n^{-c})$ decay for some $c > 0$ (with higher C meaning higher c).

- **Laplace approximation:** The rough idea of the Laplace approximation is that an integral of the form $\int e^{-nf(x)}dx$ is dominated by the part where $f(x)$ is minimized, especially if n is large. This leads to the idea of approximating $f(x)$ near its minimizer using a second-order Taylor expansion.

  In the context of sums of independent random variables, integrals like $\int e^{-nf(x)}dx$ arise from using characteristic functions, and we can use this idea to strengthen the Chernoff bound $e^{-n\psi_X^*(\epsilon)}$ by multiplying it by $\Theta\big(\frac{1}{\sqrt{n}}\big)$. A variant called the *saddlepoint approximation* gives remarkably accurate (and easy to compute) approximations, and unifies the regimes of small, moderate, and large deviations.

- **Local limit theorem:** Continuing with i.i.d. sums, with the central limit theorem being suited to deviations of $\frac{1}{\sqrt{n}}$ from the mean, one may wonder about even smaller deviations like $\frac{1}{n}$. The *local*

limit theorem says that, under suitable conditions, probabilities this close to the mean are also well-approximated by what a Gaussian would give. For example, if $Z = X_1 + \ldots + X_n$ with each X_i being ± 1 with probability $\frac{1}{2}$ each, then (when n is even) $\mathbb{P}[Z = 0] \sim \mathbb{P}[-1 \leq N(0, n) \leq 1] \sim \frac{2}{\sqrt{\pi n}}$.

- **Poisson approximations:** Binomial and multinomial distributions are ubiquitous but not always the nicest to analyze, and Poisson approximations to them are often more convenient to work with. An example result along these lines is that $\text{Bi}(n, p)$ and $\text{Poisson}(np)$ distributions differ in total variation norm by $O(p)$, meaning the two distributions are close when $p \ll 1$.

4 Computational Complexity Theory

By far the most widely-considered complexity classes are P and NP (and accordingly NP-hard / NP-complete). Very briefly, the classes P and NP concern decision problems (those with YES/NO answers). Problems in P are those that can be *solved* in polynomial time, whereas problems in NP are those that whose YES answers can be *verified* in polynomial time given a suitable "certificate" (e.g., for constraint satisfaction, the certificate is the Boolean assignment, and then it is easy to verify that all constraints are satisfied). A problem is *NP-hard* if solving it in polynomial-time would imply solving all problems in NP in polynomial time, and a problem is *NP-complete* if it is both NP-hard and in NP.

Some other ones to be aware of are outlined follows:

- For problems like optimization, we can talk about *approximately solving* the problem, and this is formalized by the notions of (Fully) Polynomial Time Approximation Scheme ((F)PTAS):
  - For **PTAS**, the requirement is that for any $\epsilon > 0$, there exists $k > 0$ for which some $O(n^k)$-time algorithm returns a solution whose value is within a multiplicative $1 \pm \epsilon$ factor of optimal. The value of k may have arbitrarily dependence on ϵ, e.g., $k = \frac{1}{\epsilon}$ or even $k = \exp(1/\epsilon)$.
  - For **FTPAS**, the requirement is still getting within a $1 \pm \epsilon$ factor of optimal, but with the stricter requirement of runtime *polynomial in both n and $\frac{1}{\epsilon}$* (e.g., $O\big(n^7(1/\epsilon)^{10}\big)$.
- There are subtle differences between being *polynomial in the number of input bits* vs. *polynomial in the (integer) sizes of the input values (and the number of such values).* Algorithms attaining the latter property are said to run in **psuedo-polynomial time** (e.g., dynamic programming for knapsack).
  - Related to this, if a problem maintains the NP-completeness property even when numerical values are constrained to be at most polynomially large (with respect to the number of input bits), it is said to be **strongly NP-complete**.
- Randomized algorithms can offer more flexibility than deterministic ones, and the class **BPP** is the probabilistic counterpart to P that allows the algorithm to be wrong with a certain probability.
- **PSPACE** and **EXPSPACE** constrain the storage space used by the algorithm (to be at most polynomial or at most exponential), rather than the runtime.
- Classes like **ETH** and **SETH** allow for *fine-grained* complexity analyses, meaning they can give statements like "attaining $O(n^{2-\epsilon})$ runtime is hard" instead of the much cruder notion of polynomial vs. higher than polynomial.

- There are certain problems that are not known to be NP-complete but are conjectured to be, such as **Unique Games**, and this forms the basis for establishing other algorithms to be at least as hard as the conjectured hard one.
- **(Relevant course: CS5230 Computational Complexity)**

5 Constraint Satisfaction Problems

- The **Constraint Satisfaction (SAT)** problem (in disjunctive normal form) consists of n Boolean variables $x_1, \ldots, x_n$ (0 = FALSE, 1 = TRUE) and m logical *clauses* $c_1, \ldots, c_m$ given by the "OR" operation applied to a set of variables and/or complements of variables. Here are some examples of clauses:

$$c_1 : x_1 \vee x_2 \vee x_3$$
$$c_2 : x_2 \vee \overline{x_5} \vee \overline{x_7} \vee x_9,$$

  The problem asks whether there is any assignment to $x_1, \ldots, x_n$ such that all clauses are true, i.e., $c_1 \wedge \ldots \wedge c_m$ holds. This was the first problem shown to be NP-complete.
- The 3SAT problem is a special case of SAT in which each clause can only consist of at most 3 variables. This may sound easier than the general SAT problem, but in fact it is possible to reduce SAT to 3SAT, so even 3SAT is NP-complete. 3SAT is the most common NP-complete problem used for subsequent reductions to prove hardness results. Ideas from constraint satisfaction also form the basis for algorithms and theory in a variety of topics throughout computer science and beyond.
- Even if it's impossible to satisfy every constraint, we may still be interested in satisfying *as many as possible*. This leads to the **Maximum Satisfiability (MAXSAT)** problem – find Boolean assignments to $x_1, \ldots, x_n$ to maximize the number of clauses satisfied
- Even though SAT and MAXSAT are "hard" problems, it should be kept in mind that NP-hardness is a worst case notion. Practical SAT solvers do a remarkably good job of solving practical (non-worst-case) instances of very large size.
- See `https://www.youtube.com/watch?v=zqNEtGfGGmA` for a "TCS toolkit" style introduction.
- **(Relevant course: CS4269/CS5469 Fundamentals of Logic in Computer Science)**

6 Sketching and Streaming

- The rough idea of sketching is to take a "large" object (e.g., a long vector) and "sketch" it down to a smaller object (e.g., a short vector) that still retains (most of) the relevant information.
- One of the most famous examples of sketching is the *Johnson-Lindenstrauss (JL) lemma*: Given points $\mathbf{x}_1, \ldots, \mathbf{x}_m$ in $\mathbb{R}^n$ (think of n as large) if we construct a suitably-normalized i.i.d. Gaussian matrix $\mathbf{A} \in \mathbb{R}^{t \times n}$ and form $\mathbf{z}_1, \ldots, \mathbf{z}_m$ in $\mathbb{R}^t$ with $\mathbf{z}_i = \mathbf{A}\mathbf{x}_i$ (think of t as relatively small), then we have with

high probability that *pairwise distances are roughly preserved*, i.e.,

$$(1-\epsilon)\|\mathbf{x}_i - \mathbf{x}_j\| \leq \|\mathbf{z}_i - \mathbf{z}_j\| \leq (1+\epsilon)\|\mathbf{x}_i - \mathbf{x}_j\|, \quad \forall i, j,$$

as long as $t \geq C\frac{\log m}{\epsilon^2}$ for a suitable constant C. This roughly means that for any task that only depends on pairwise distances (e.g., certain clustering algorithms), we can expect similar performance by working with the $\mathbf{z}$'s instead of the $\mathbf{x}$'s.

- A limitation of the JL lemma is that $\mathbf{A}$ is a dense matrix, so it can be expensive to store all $t \times n$ of its values and to perform multiplications using it. To overcome this, sketching methods often seek that $\mathbf{A}$ is *sparse* (mostly 0s) and/or has other properties that permit fast computation and low storage.
- Two popular examples are *Bloom filters* and *count-min sketch*, but we will not go into detail here. See `https://arxiv.org/abs/1411.4357` for a survey of the topic.
- A closely related notion is *streaming*, where the elements of a long vector (or other object) $\mathbf{x}$ arrive one-by-one, but $\mathbf{x}$ is so long that we can't hope to store every element. Ideas from sketching allow us to store a "compressed version" that retains the information that we need.
- **(Relevant course: CS5234 Algorithms at Scale)**

7 Hashing

- Roughly speaking, a hash function is a function that takes as input a (possibly large) "object" and maps it to a value in a limited range, with the hope that distinct inputs "usually" get mapped to distinct outputs. For example, given two student numbers, the final 3 digits will usually be different, so this could potentially be used as a hash function.
- Two example uses of hash functions are as follows:
  (i) We are storing a database of such "objects" and we use the hash function to quickly look up where they are stored.
  (ii) Suppose that we want to see whether our local 10GB file is up-to-date with a server's version. Instead of downloading the 10GB file and checking bit-by-bit, we can just request a 64-bit hash value (say) from the server, and compare it against the same hash function applied to our local file. If the hashes match (and the hash function is well-chosen), we can be extremely confident that the files are the same. If the hashes don't match, we can be certain that the files differ.
- Sometimes hash functions are designed to have low probabilities of collisions when the input objects are uniformly random (e.g., every student number is a uniformly random 7-digit number). However, this assumption is often violated in practice (including in the case of student numbers).
- An alternative perspective is to consider *worst-case inputs* but let the *hash function itself be random*, and show that it "performs well" with high probability.
- Suppose that the inputs lie in some set $\mathcal{X}$ and the output set $\mathcal{Y}$ of the hash function has size $|\mathcal{Y}| = m$ (e.g., $\mathcal{Y} = \{1, \ldots, m\}$). A "fully random" hash function $h : \mathcal{X} \to \mathcal{Y}$ would be one that independently

assigns every element of $\mathcal{X}$ a uniformly random value in $\mathcal{Y}$. This would give excellent properties in terms of avoiding collisions, but it is impractical because there is no efficient way to store such an h.

- Fortunately, it can be much easier to obtain less restrictive properties that still suffice for proving desirable mathematical results:
  - *Universality*: For any $x, x' \in \mathcal{X}$, we have $\mathbb{P}[h(x) = h(x')] \leq \frac{1}{m}$. (We can also relax this by replacing 1 by a higher constant, giving *approximate universality*.)
  - *k-wise Independence*: For any k inputs $x_1, \ldots, x_k$ and any output indices $i_1, \ldots, i_k$, we have $\mathbb{P}[h(x_1) = i_1, \ldots, h(x_k) = i_k] = \frac{1}{m^k}$ (i.e., same as the fully random case). For $k = 2$ this is called *pairwise independence* or *strongly universality*.
- Here is an example of a strongly universal hash function. Suppose that $\mathcal{X} = \{0, 1, \ldots, p-1\}$ for a (large) prime number p. For now suppose that $\mathcal{Y} = \mathcal{X}$, but we will relax this below. If we let a be uniformly random in $\{1, \ldots, p-1\}$ and b be uniformly random in $\{0, 1, \ldots, p-1\}$, then it is fairly straightforward to show that
$$h_0(x) = ax + b \pmod{p}$$
is pairwise independent. (We avoid $a = 0$ since multiplying by 0 would always give 0.) Then to get a hash function with a smaller output size m, we can simply let $h(x) = h_0(x) \pmod{m}$; this "essentially" maintains the strong universality property, up to minor rounding issues.
  - Another example based on sequences of bits (rather than integers) will be given in the next section.
- Observe that the hash function is fully specified by a and b, so we only need $2\lceil \log_2 p \rceil$ bits of storage. In contrast, directly storing a fully random hash function would require storing all p hash values, which amounts to around $p \log_2 m$ bits. So there is a major difference of $O(\log p)$ vs. $\Omega(p)$.
- **(Relevant course: CS5330 Randomized Algorithms)**

8 Derandomization and Pseudorandomness

- Derandomization of the process of taking a randomized algorithm and adapting it into a deterministic one (while preserving guarantees such as correctness or approximate correctness), or more generally, one that at least uses less randomness.

  From a practical view, deterministic algorithms may be favored due to being more consistent/predictable, and from a theoretical view, we are often very interested in distinguishing between what is possible probabilistically vs. deterministically (or using limited randomness, as random bits are often considered a limited resource).
- We saw an example of derandomization for MAXCUT in the Probabilistic Method lecture, and similar ideas can be applied to other problems.
- In general, derandomization is simple if the number of random bits used is "small". For example, if a randomized algorithm uses 10 random bits, then we can just search over all 2^{10} combinations and take the correct/best one (assuming there's a way to check for correctness or what's "best"). More generally, if the input size is n and only $O(\log n)$ random bits are used, then we can search over all combinations in polynomial time.

- This consideration leads to the notion of a *pseudorandom generator* (PRG): Letting $\mathcal{F}$ be a class of Boolean functions with n inputs (i.e., $f : \{0,1\}^n \to \{0,1\}$), we say that a function $G : \{0,1\}^\ell \to \{0,1\}^n$ is an ϵ-PRG with respect to $\mathcal{F}$ if:

$$\left|\mathbb{P}_S[f(G(S)) = 1] - \mathbb{P}_X[f(X) = 1]\right| \leq \epsilon, \quad \forall f \in \mathcal{F},$$

  where S is uniform on $\{0,1\}^\ell$ and X is uniform on $\{0,1\}^n$. We call ℓ the *seed length*, and according to the previous dot point, we ideally want (i) a low seed length such as $\ell = O(\log n)$, and (ii) G itself to have low storage and computation requirements.

  - Intuition: $\mathcal{F}$ represents a class of "tests" that someone is trying to perform to check whether the bits are truly random or not. The above condition ensures that none of these tests are able to confidently say YES or NO. The "richer" $\mathcal{F}$ is (e.g., containing all functions implementable in time $O(n^c)$ and letting c increase), the stronger the guarantee of the above equation is.

- PRGs also tie into the idea of k-wise independence outlined in the Hashing section above. If we can show that a randomized algorithm works well when the entries of X are k-wise independent, then we can exploit the fact that there exist functions $G : \{0,1\}^\ell \to \{0,1\}^n$ producing k-wise independent output bits (with each bit being uniform on $\{0,1\}$) under the scaling $\ell = O(k \log n)$. In particular, we get the desired $\ell = O(\log n)$ scaling if k is a constant (e.g., 2, 4, or 32).

  - For example, if $n = 2^{\ell-1}$, then we can get pairwise independence by taking a uniformly random bit string $[S_1, S_2, \ldots, S_\ell]$ and multiplying it by an $\ell \times n$ matrix whose columns contain all non-zero length-ℓ binary strings. (Proof omitted.) In this case, we have $\ell = 1 + \log_2 n$.
  - More advanced extensions of this example use tools from coding theory (e.g., Reed-Solomon).

- See Chapter 4 of `https://ccanonne.github.io/teaching/COMPx270` for more on derandomization

- See `https://www.youtube.com/watch?v=3lCmIM3lH8Y` (and the further lectures following it) for a "TCS toolkit" style introduction to psuedorandomness.

- **(Relevant course: CS5330 Randomized Algorithms)**

9 Graph Theory and Algorithms

Graphs are ubiquitous in algorithm design and theoretical computer science, and there are many relevant sub-topics:

- Random graph theory:

  - This broadly concerns what properties emerge when we generate graphs randomly, with a particularly common random distribution being the Erdös-Rényi model: For each pair (i, j) of nodes, independently include an with probability p.
  - The types of questions asked include: For which (p, n) does the graph become connected? When do cliques (fully-connected sub-components) of certain sizes start to appear? What is the chromatic number of the graph? etc.

  - These questions often permit remarkably precise answers, with rapid *phase transitions* between some property (e.g., connectivity) being observed with probability nearly 0 and probability nearly 1 as p increases.

- Spectral graph theory:
  - Broadly speaking, spectral graph theory allowed us to understand properties of graphs using notions from linear algebra, particularly eigenvalues and eigenvectors.
  - An immediate connection between graphs and matrices is that we can represent a graph via its *adjacency matrix* $\mathbf{A}$ (with $A_{ij} = 1$ if i and j are connected). But more commonly a related matrix called the *Laplacian* is used: $\mathbf{L} = \mathbf{D} - \mathbf{A}$, where $\mathbf{D}$ is a diagonal matrix whose i-th diagonal entry is the degree of of the i-th node. (There are also normalized variants, e.g., for d-regular graphs this is done by dividing by d to get the *normalized Laplacian* $\mathbf{I} - \frac{1}{d}\mathbf{A}$.)
  - One example property relating eigenvalues to graph properties is as follows: The multiplicity of the eigenvalue 0 is exactly equal to the number of connected components in the graph.
  - A famous result called *Cheeger's inequality* refines this idea, and relates a natural measure of how "well-connected" the graph is to the second-smallest eigenvalue of $\mathbf{L}$.
  - These ideas lead to algorithms that work with the eigenvalues of $\mathbf{L}$, e.g., for the problem of *community detection* where we want to cluster the nodes into groups with high intra-connectivity but low inter-connectivity. (The eigenvector associated with the second-smallest eigenvalue can tell us the community structure.)
  - See `https://www.youtube.com/watch?v=Nv3EaRL60ww` for an introduction to spectral graph theory, and `https://www.youtube.com/watch?v=gwxuipf-9IQ` for some TCS toolkit style lectures.

- Graph sparsifiers: The goal is to take a "dense" graph (many edges) and use it to produce a "sparse" graph that maintains certain properties of interest (e.g., cut properties or spectral properties). This is similar in spirit to sketching; using the "sparsified" graph can alleviate downstream requirements of storage, computation, etc.

- Bounded treewidth algorithms:
  - Algorithms that operate on graphs frequently work efficiently and correctly when the graph is a tree, but may fail on more general graphs.
  - The *treewidth* is, very roughly speaking, a measure of "how far" a graph is from being a tree, and significant effort has been put into *bounded treewidth algorithms* that work beyond the case of trees while still maintaining the desirable properties.
  - See `https://www.youtube.com/watch?v=kEnDGTwSDXY` for a TCS-toolkit style introduction.

- **(Relevant course: CS5234 Algorithms at Scale)**

10 Cryptography

- The theory and tools of cryptography extend far beyond the most well-known RSA strategy, and there are close connections to other areas of computer science such as hashing, pseudorandomness, information theory, and others.

- A typical (shared key) setup is as follows:
  - A sender (Alice) would like to send a message m to a receiver (Bob)
  - An eavesdropper (Eve) has access to Alice's output, and Alice and Bob want the message to be kept secret from Eve (*even when Eve knows the communication protocol Alice and Bob are using*)
  - To have some hope of doing this, we assume that Alice and Bob have access to a (random) *shared key* K, but Eve does not. So Alice sends some "codeword" $c = \mathrm{Enc}(m, K)$, Bob recovers the message via some $\mathrm{Dec}(c, K)$, and yet we want c alone (being visible to Eve) to reveal little or nothing about m when K is unknown.
- A notion called *perfect secrecy* requires that the randomness of K gives $\mathbb{P}[\mathrm{Enc}(m_0, K) = c] = \mathbb{P}[\mathrm{Enc}(m_1, K) = c]$ for any two messages m_0, m_1. This is attainable via a technique called *one-time pad*, but the number of bits of randomness in K needs to be the same as the number of bits used to represent m, which is typically impractical.
- Much of the foundations of cryptography concern relaxing the perfect secrecy requirement in two ways:
  - We don't require the two probabilities to be identical, but instead allow them to have a "negligible" difference (e.g., $n^{-\omega(1)}$).
  - We don't require an *arbitrary* adversary to be unable to distinguish messages, but instead only a *computationally bounded adversary* (formally, PPT – probabilistic polynomial-time).

  According to the second of these, it may still be *mathematically possible* for an adversary to learn something about the message, but we design the system so that it is *computationally infeasible* for them to do so.
- See `https://www.youtube.com/watch?v=Ptl7pjDmQjk` (and the further lectures following it) for a TCS-toolkit style introduction to this topic.
- **(Relevant course: CS4230/CS5430 Foundations of Modern Cryptography)**

11 Other Mathematical Tools

The list of mathematical tools that are useful in computer science is virtually endless, so we won't go into further detail, but briefly mention some others:

- Random matrix theory (e.g., eigenvalues of Gaussian random matrices)
- Concentration bounds for sums of random matrices (e.g., see `https://arxiv.org/abs/1004.4389`)
- Stochastic processes (e.g., Gaussian process);
  (Relevant course: MA5249 Stochastic Processes and Algorithms)
- Ordinary/partial differential equations (e.g., for understanding dynamical systems)
- (etc.)