CS5275 Lecture 12: Other Topics

Jonathan Scarlett

December 6, 2024

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.

[TODO: More links to https://ccanonne.github.io/teaching/COMPx270]

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 $\|\mathbf{x}\|_p = \left(\sum_{i=1}^n |x_i|^p\right)^{1/p}$ (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 amount of dirt we can move 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 (among others), 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 asymptotic normality result:

$$\sup_x \left|F_n(x) - \Phi(x)\right| \leq \frac{C\rho}{\sigma^3\sqrt{n}},$$

where C is a (small) 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) $\mathbb{P}[Z = 0] \sim \mathbb{P}[-1 \le N(0,n) \le 1] \sim \frac{2}{\sqrt{\pi n}}$.

- **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 have 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" $c = \text{Enc}(m, K)$, Bob recovers the message via some $\text{Dec}(c, K)$, 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 $\mathbb{P}[\text{Enc}(m_0, K) = c] = \mathbb{P}[\text{Enc}(m_1, K) = c]$ 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.)