Advanced Algorithms

Lecturer: Ben Lee Volk (benlee.volk@runi.ac.il)

Details

- Slides, problem set, forum for discussions – all available on the moodle web page
- No single textbook – we'll cover portions from various books
- Final grade composition:
  - 30% homework (4-5 problem sets)
  - 70% Final exam (must get at least 60 to pass)

Problem sets

- Homework grade will be calculated as average of the best n-1 grades among the n problem sets
- OK to collaborate with others and consult external sources but please **write your own solutions**
- I can’t stop you from using AI chatbots but their solutions are often wrong, so it’s better to try to solve it yourself
- Submit a PDF file. +5 point bonus for typed submissions.

Course Goals

- Getting acquainted with a large algorithmic toolbox
- Introduction to different advanced models and techniques
- A better sense of how to model problems you encounter as well-known algorithmic problems
- Practice in reasoning formally about computation and writing arguments and proofs
- Fun?

Prerequisites

- This is an advanced class (mostly for grad students)
- Basic algorithms class (graph theory, elementary graph algorithms)
- Linear Algebra (vector spaces, dimensions, matrices, ranks, bases, linear independence)
- Probability (random variables, expectations, independence)
- Some (but not a lot) familiarity with complexity theory (asymptotic running times, big O notation)
- “Mathematical maturity” (ability to understand definitions, read and write proofs)
- You’re not expected to remember every little details from those classes! I’ll remind you necessary facts

Some topics

- Randomized algorithms
- Algebraic Algorithms
- Approximation Algorithms
- Linear Programming and applications
- Other advanced topics, time permitting

Each of these topics is a vast area that deserves its own course!

Algorithm design goals

- Correctness
- Efficiency
- Simple (if possible)
- Easy to implement

Each of the above can be addressed from a theoretical or practical viewpoints (this class is a theoretical class).

Measuring complexity

Measuring the running time of an algorithm on a computer (in seconds) isn't a good idea.

We want a stable measure, independent of a specific implementation

The idea (which you already saw) is to measure **asymptotic complexity** as a function of the input length

RAM model of computation

- … not worth our time to define formally
- But we all have a sense of what it means!
- Every “basic operation” takes one time step
- Memory can be directly address through pointers (unlike one-tape Turing machines)
- Running time = number of steps
- Space usage = number of memory cells used

Measuring Complexity

It's usually easiest to measure **worst case complexity**.

It's (quite often) not the most relevant metric for real world apps!

We can also measure **average case** complexity and other notions – won't be the focus of this class.

We say an algorithm A has time complexity $T(n)$ if for every input of length n, A runs at most $T(n)$ steps.

We'll sometime mention **space complexity** (but it's not the main focus of the class)

Asymptotic Complexity

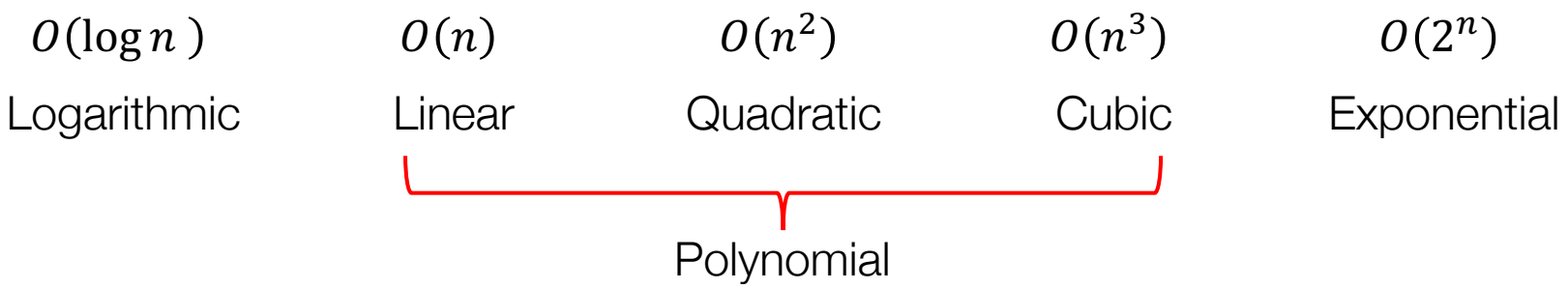
We usually only care about the **growth rate** of $T(n)$ and not the exact expression:

- The exact expression may be complicated to compute
- And it may depend on the exact model
- And it may depend on the encoding of the input

We say $f(n) = O\big(g(n)\big)$ if there's a constant c s.t. $f(n) \leq c \cdot g(n)$

Growth rates

- Even by ignoring constant factors, we can get an excellent idea of whether a given algorithm will be able to run in a reasonable amount of time on a problem of a given size
- The big O notation and worst-case analysis are tools that greatly simplify our ability to compare the efficiency of algorithms



Randomized Algorithms

Book: “Randomized Algorithms” by Motwani and Raghavan

Do you remember Quicksort?

Quicksort is a sorting algorithm that works as follows.

Input: a set $X = \{x_1, \ldots, x_n\}$

Output: the elements of X is ascending order

1. Pick a "pivot" element $p \in X$
2. Set $Y =$ all elements that are at most p, and $Z =$ all others
3. Recursively sort Y, Z, return Y sorted, then p, then Z sorted

What's the running time?

- Depends on how the pivot is picked
- There's a way to do the other steps in time $O(n)$
- If p is the median, there are $\frac{n}{2}$ elements in Y and Z, and we get the recursive equation $T(n) = 2T\left(\frac{n}{2}\right) + O(n)$ whose solution is $T(n) = O(n \log n)$
- If p is an element such that there are $n/3$ in Y and $2n/3$ in Z, similar guarantees
- Same with $n/10$ and $9n/10$

How to pick the pivot?

- Option 1: arbitrarily (pick the first element in the array)
  - "Usually" works, but we can engineer examples where this would take time $O(n^2)$
- Option 2: there's an $O(n)$ time algorithm to pick the median
  - But who remembers how it goes? We learned this a long time ago
- Option 3: pick p at random!
  - Huh?

Randomized Algorithms

- We assume our algorithm has unlimited supply of **unbiased, independent** random coins
- The output of the algorithm is now a **distribution** on outputs
- The running time of the algorithm is now a **distribution**
- We still consider **worst case analysis**: we want guarantees for **every** input and not for “most” inputs (we’ll touch on this point further later)

Back to Quicksort

- One can show that **in expectation** (in fact, with high probability), if we pick the pivot at random, the running time will be $O(n \log n)$
  - Analysis isn't hard but we'll skip it: See [Motwani-Raghavan]
- This is true for **every** input array: you can't engineer an input which would make the algorithm run slowly
- There **exist** outcomes for the random coin tosses that would make the algorithms run slowly (that's why we can't just pick arbitrarily), but "most" coin tosses are fine

Finding hay in haystack

- In many algorithmic problems, we can show that “most” choices are good, but it’s hard to come up with a deterministic procedure that always produces a good choice
- In these circumstances, a random choice is useful: it produces a good choice (with high probability) **for every input**
- We’ll soon make this discussion more formal

Finding Prime Numbers

- Goal: given n, find a prime number with n digits
  - Useful for cryptography. In RSA public-key encryption, the private key is a pair of n-bit primes p, q and the public key is $p \cdot q$
- Deterministic algo: Start with $\underbrace{100\ldots.0}_{n}$ and check one by one
- For all we know, this takes time which is exponential in n 10^n
- Randomized algorithm: pick a **random** integer with n digits, and check for primality (this can be done efficiently)
- What's the probability of hitting a prime number?

Prime Number Theorem

- **Theorem:** the fraction of primes among all n-digit numbers is approximately $1/n$.
- Doesn't look **that** good…
- But what if we repeat the same algorithms n^2 times, each time with independent random coins?
- Let E_i be the event that the i-th run didn't yield a prime number. Then $\Pr[E_i] \approx 1 - 1/n$.
- Probability that all attempts fail is

$$\Pr[E_1 \cap E_2 \cap \cdots \cap E_{n^2}] = \prod_{i=1}^{n^2} \Pr[E_i] \leq \left(1 - \frac{1}{n}\right)^{n^2} \leq e^{-n}$$

- Failure probability is exponentially small!

למת המספרים הראשוניים

נסמן ב- $\pi(n)$ את כמות המספרים הראשוניים שקטנים או שווים ל- n.

נטען כי $\pi(n) = \frac{n}{\log(n)}$.

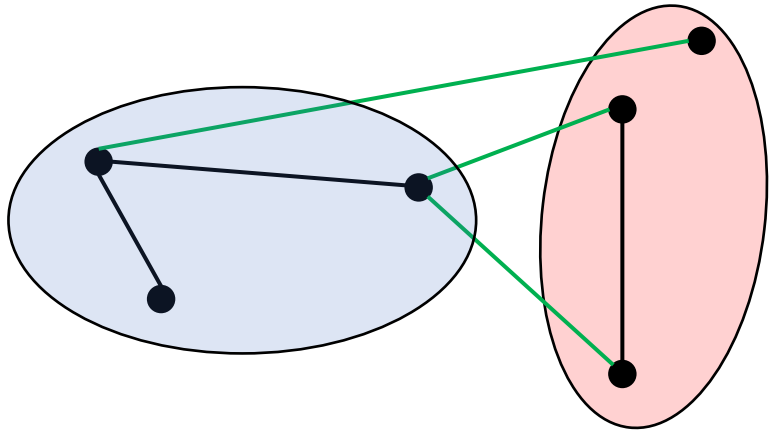
לשם כך, נסתכל על $\frac{1}{\log(n)}$ את ההסתברות שמספר $n, \ldots, 1$ הוא מספר ראשוני.

Different Kinds of Algorithms

- In the Quicksort example: output was always correct (a sorted array), running time was $O(n \log n)$ **in expectation**.
  - Also called "Las Vegas Algorithm" – I won't use this name
  - A better name is "zero error" – these algorithms never err
- In the finding primes example: output can be wrong with small probability, running time is polynomial **in the worst case**
  - Also called "Monte Carlo Algorithm" – I won't use this name
  - A better name is "bounded error" – the error probability is small
- In the homework you'll prove that a zero error algorithm implies a bounded error algorithm

Another example: cuts in graphs

- Let $G = (V, E)$ be an undirected (multi-)graph
- A **cut** in G is a partition of the vertices into two disjoint non-empty sets $V = A \cup B$
- The **size** of the cut is the number of edges crossing the cut: i.e. edges (u, v) such that $u \in A$ and $v \in B$



Minimum and Maximum cut

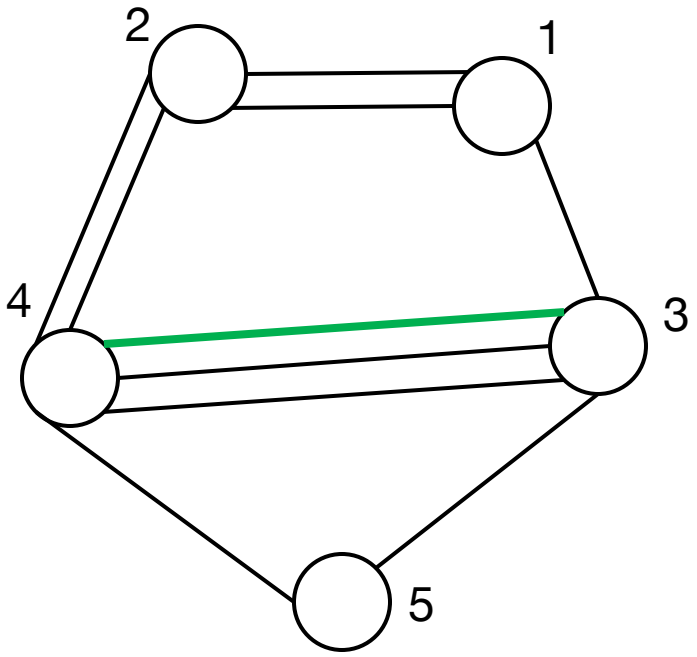
- Finding the maximum cut is an NP hard problem
- Finding the minimum cut is efficiently solvable using max flow algorithms you learned in the basic algorithms class (“min-cut max-flow theorem”)
- We’ll see a much simpler and more efficient **randomized** algorithm that finds the minimum cut in a graph
- Max cut will return towards the end of the class!

Karger's Min-Cut Algorithm

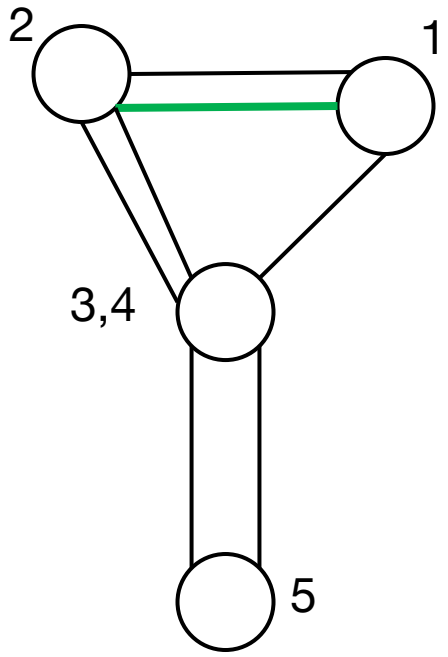
1. While the number of vertices in G is more than 2:
   1.1. Pick a **random** edge $e = (u, v) \in E$
   1.2. **Contract** e: marge u, v to a new node $\{u, v\}$, delete all self loops obtained in this process.
2. Return the cut corresponding to the remaining two vertices

Running example:

Iteration 1

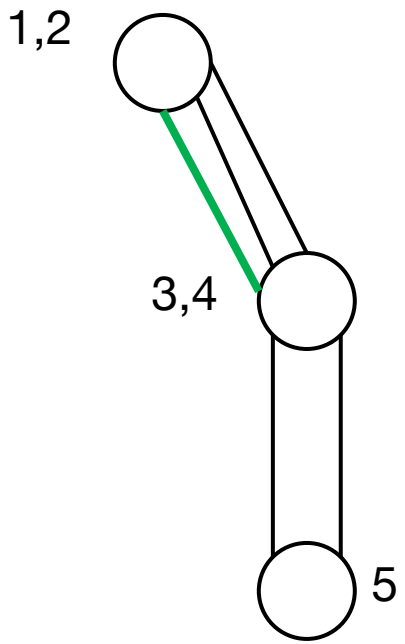


Iteration 2

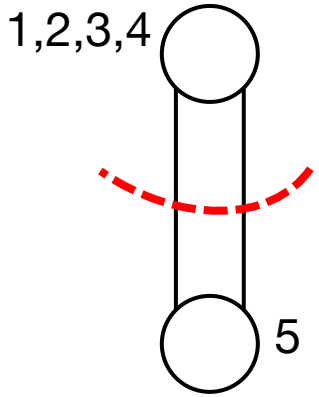


Running example:

Iteration 3

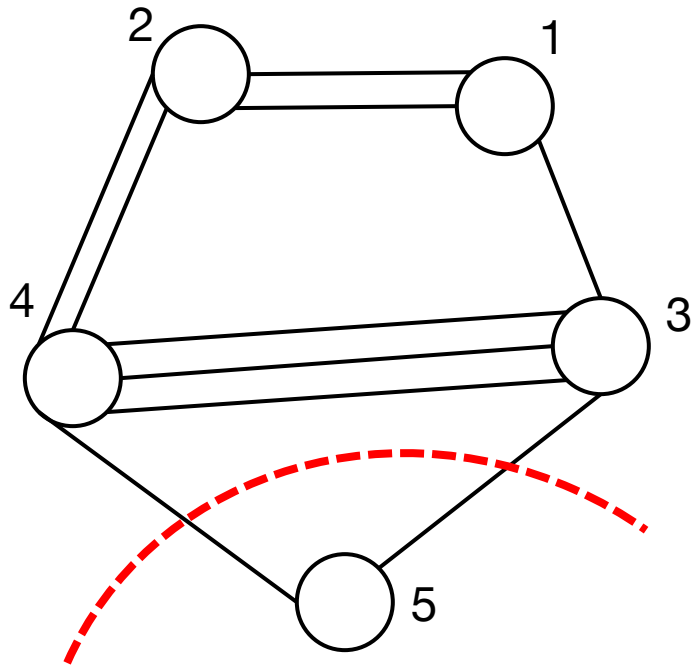


Iteration 4



Stop: only 2 vertices remain

Output



Min-Cut Algorithm Analysis

- Let $A \cup B$ be a minimum cut in G
- The algorithm returns $A \cup B$ if none of the edges crossing this cut are contracted during its run
- Intuition: since $A \cup B$ is a "small" cut and the algorithm picks a random edge, it's somewhat likely the algorithm will never pick an edge crossing it
- Let k be the number of edges crossing the cut
- We want to estimate the probability that on a given iteration, the algorithm **doesn't** pick one of these edges

Min-Cut Algorithm Analysis

Claim: For every vertex v in G, $\deg v \geq k$.

Proof: Otherwise, $\{v\}, V \setminus \{v\}$ is a smaller cut.

Corollary: $|E| = \frac{1}{2}\sum_{v \in V} \deg v \geq \frac{kn}{2}$ (where $n = |V|$)

Therefore, the probability of **not** picking an edge that crosses that cut $A \cup B$ in the first iteration is

$$1 - \frac{k}{|E|} \geq 1 - \frac{2}{n}$$

$$1 - \frac{k}{m} \geq 1 - \frac{k}{\frac{kn}{2}} = 1 - \frac{2}{n}$$

Min-Cut Algorithm Analysis

Let E_i denote the event that the cut $A \cup B$ survives the i-th iteration.

Conditioned on $E_1, \ldots, E_i$, what's the probability of the cut surviving the $(i+1)$-th iteration?

- The number of vertices is now $n - i$
- The size of the cut is still k
- The minimum degree is still $\geq k$ (otherwise we would have had a smaller cut in G)

So,

$$\Pr[E_{i+1} | E_1, \ldots, E_i] \geq 1 - \frac{2}{n-i}$$

$$1 - \frac{k}{m} \leq 1 - \frac{k}{\frac{k(n-i)}{2}} = 1 - \frac{2}{n-i}$$

Min-Cut Algorithm Analysis

The probability that the cut survives **all** contractions is

$$\Pr[E_1 \cap E_2 \cap \cdots \cap E_{n-2}] = \prod_{i=0}^{n-3} \Pr[E_{i+1} | E_1, \ldots, E_i]$$

$$\geq \prod_{i=0}^{n-3} \left(1 - \frac{2}{n-i}\right) = \prod_{i=0}^{n-3} \frac{n-i-2}{n-i} = \frac{2}{n(n-1)}.$$

Is this bad?

Min-Cut Algorithm Analysis

Repeat the algorithm n^3 times, each time with fresh, independent random coins.

Pick the minimal cut found in all these runs.

The probability $A \cup B$ is never picked is exponentially small in n

Polynomial Identity Testing

Give two polynomials $f(x_1, \ldots, x_n)$ and $g(x_1, \ldots, x_n)$, are they identical?

Easy if both polynomials are given as list of coefficients

But there are more succinct ways to represent polynomials:

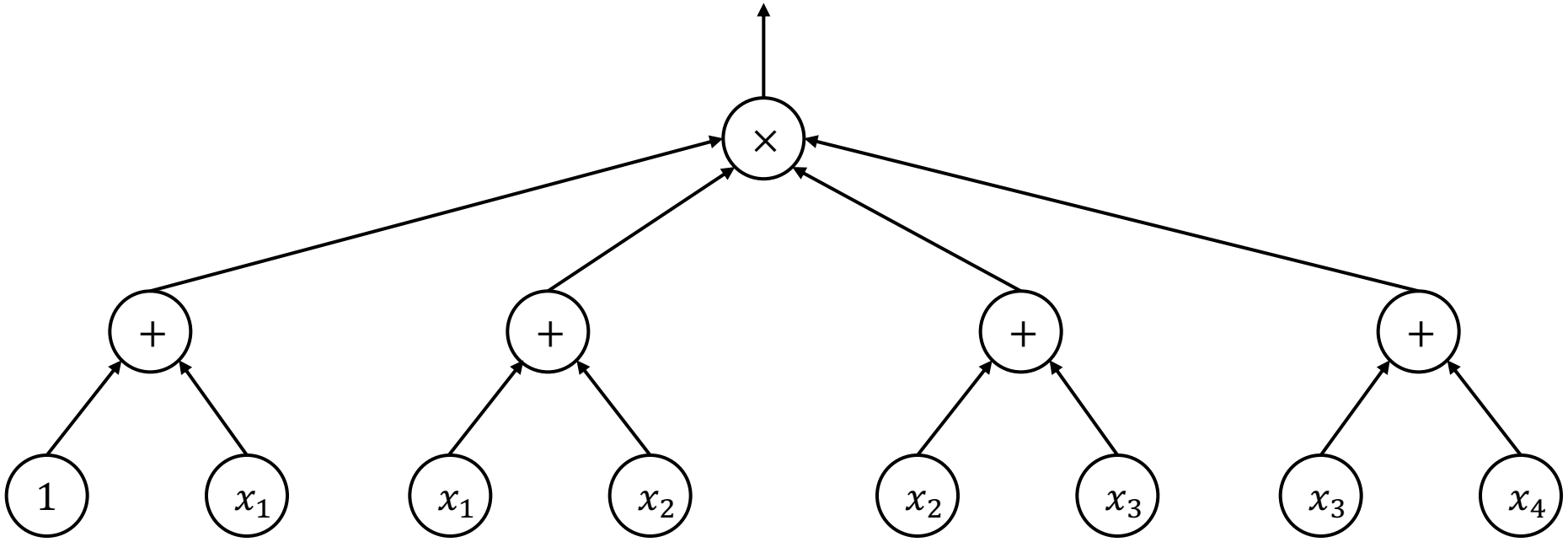
Example:

$$(x-5)^3 \cdot (x+2)^2 + y^5 \cdot (1+x+x^3)^4$$
$$\stackrel{?}{=} x^{12}y^5 + 4x^{10}y^5 + 4x^9y^5 + 6x^8y^5 + 12x^7y^5 + 10x^6y^5 + 12x^5y^5 + x^5$$
$$+ 13x^4y^5 + 19x^4 + 8x^3y^5 + 139x^3 + 6x^2y^5 + 485x^2 + 4xy^5 + 800x$$
$$+ y^5 + 500$$

Deterministic Algo: expand both sides and compare (takes exponential time)

Polynomial Identity Testing

More formally: we represent the input polynomials as **formulas**:



Polynomial Identity Testing

- Small formulas can represent polynomials with exponentially many non-zero monomials, e.g., $\prod_{i=1}^{n}(1+x_i)$ ← single term, like $3x^2y^3z^5$
- Given f and g we can equivalently define $h = f - g$ and check whether $h = 0$.
- Equivalent formulation of PIT: given a formula F, decide if F computes the identically zero polynomial
- Idea: plug "random numbers" $\alpha_1, \ldots, \alpha_n$ to the and check if $F(\alpha_1, \ldots, \alpha_n) = 0$
- This computation can be done in polynomial time

Polynomial Identity Testing

- What's the probability that $F(x_1, \ldots, x_n)$ evaluates to 0?
- A nonzero degree d univariate polynomial has at most d roots
- Not true even for bivariate polynomial (e.g., $x - y$)

Schwartz Zippel Lemma: Let f be a nonzero n-variate degree-d polynomial with rational coefficients. Let $S \subseteq \mathbb{Q}$ be a finite set.

$$\Pr_{\alpha_1, \ldots, \alpha_n \in S}[f(\alpha_1, \ldots, \alpha_n) = 0] \leq \frac{d}{|S|}$$

where $\alpha_1, \ldots, \alpha_n$ are picked uniformly at random from S.

Polynomial Identity Testing

Schwartz Zippel Lemma: Let f be an n-variate degree-d polynomial with rational coefficients. Let $S \subseteq \mathbb{Q}$ be a finite set.

$$\Pr_{\alpha_1,\dots,\alpha_n \in S}[f(\alpha_1, \dots, \alpha_n) = 0] \leq \frac{d}{|S|}$$

where $\alpha_1, \dots, \alpha_n$ are picked uniformly at random from S.

Remarks

- Proof: by induction on n, in HW
- Base case $n = 1$ follows immediately from the fact that a univariate polynomial has at most d roots
- A formula of size s computes a polynomial of degree at most s (by induction on the structure)

Polynomial Identity Testing

Input: A formula F of size s

Output: "Yes" if $F = 0$. "No" (with high probability) if $F \neq 0$

1. Define $S = \{1,2, \dots, 2s\}$
2. Pick uniformly at random and independently $\alpha_1, \dots, \alpha_n \in S$ and evaluate $F(\alpha_1, \dots, \alpha_n)$. Out "yes" if 0, "no" otherwise

Analysis: If $F = 0$ algorithm only answers "yes". Otherwise, by Schwartz-Zippel Lemma, $\Pr[\text{yes}] \leq \frac{1}{2}$.

How to make error probability smaller?

$\left(1 - \frac{1}{2}\right)^t < \varepsilon$

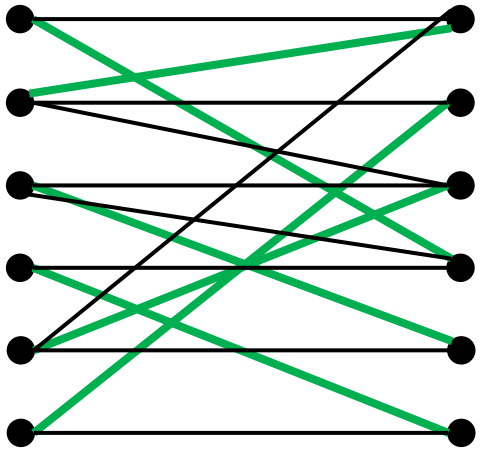
$\frac{1}{2^t} < \varepsilon$

$t \log \frac{1}{2} < \log \varepsilon \iff -t < \log \varepsilon \iff -\log \varepsilon < t$

Application: Bipartite Matching

Let G be a bipartite graph $G = (L \cup R, E)$ with $|L| = |R| = n$

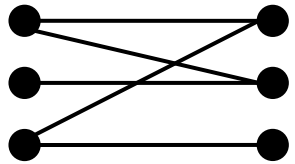
A **perfect matching** in G is a set of n vertex-disjoint edges



Bipartite Matching

- Problem: given bipartite G, decide if G has a perfect matching
- There's a deterministic efficient algorithm for this.
- Can also be reduced to the max-flow problem.
- (More natural: finding a perfect matching, if exists. We'll discuss this later)
- We'll see that this is a special case of PIT and therefore there's an efficient and simple **randomized** algorithm
- The randomized algorithm will also be **parallelizable** which is something we'll discuss later in the course

Biadjacency matrix



$$\begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

Let A be the $n \times n$ biadjacency matrix of G:

- Rows indexed by vertices in L
- Columns indexed by vertices in R
- $A_{ij} = 1$ if $(i, j) \in E$, zero otherwise

Claim: if G doesn't have a perfect matching, $\det(A) = 0$

Proof: $\det A = \sum_{\sigma \in S_n} (-1)^{\operatorname{sgn} \sigma} \prod_{i=1}^{n} A_{i,\sigma(i)}$

Each permutation σ corresponds to a potential matching

If each σ isn't a matching, there's i such that $A_{i,\sigma(i)} = 0$.

פרמוטציות

$$\sum_{\sigma \in S_n} sign \prod_{i=1}^{3} A_{i,\sigma(i)}$$

(i – מתאימים שורה; $\sigma(i)$ – עמודה)

$$\det\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} = aei - afh - bdi + bfg + cdh - ceg$$

במילים אחרות: כל פעם בוחרים איבר משורה אחרת, ועמודה אחרת. כלומר, פרמוטציה של כל איבר בשורה וטור עם כל האיברים שבשורה וטור אחרים.

פרמוטציה לדוגמא: $\sigma = [3,2,1]$

$$\Longrightarrow A_{13} \cdot A_{12} \cdot A_{11}$$

$i, \sigma(i)$

Matchings and Determinants

Is the converse also true? Perfect Matching ⇒ non-zero det?

No. Counterexample: $\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$

The problem is with the **signs**: the + and - can cancel out

Solution: define a different matrix where there are no cancellations

Symbolic Biadjacency Matrix

Given a graph G define an $n \times n$ matrix X as:

- If $(i,j) \in E$, $X_{i,j} = x_{i,j}$ (a variable)
- If $(i,j) \notin E$, $X_{i,j} = 0$

Claim: $\det X \neq 0$ (as a polynomial) iff G has a perfect matching

Proof: $\det X = \sum_{\sigma \in S_n} (-1)^{\operatorname{sgn} \sigma} \prod_{i=1}^{n} X_{i,\sigma(i)}$

We already saw each permutation is a potential matching. If indeed $X_{i,\sigma(i)} \neq 0$ for all i we get a monomial that can't be cancelled out by any other monomial

$$\begin{bmatrix} x_{1,1} & x_{1,2} \\ x_{2,1} & x_{2,2} \end{bmatrix}$$

Testing zeroness of Determinant

How to test whether $\det X = 0$ as a polynomial?

Solution: PIT. $\det X$ is a polynomial with at most n^2 variables of degree at most n.

Given any assignment to the variables, the determinant can be computed efficiently (say using Gaussian elimination or other algorithms we'll see later in the class)

Error reduction

- In all the algorithms we saw so far, it was possible to reduce the error and make it tiny without paying too much in the running time
- For all practical purposes, an error with probability 10^{-10} is as good as no error at all
- We will now prove the general conditions under which this can be done
- For simplicity we'll focus on decision problems (algorithms that answer "yes" or "no", also called "languages" in a complexity theory class)

Error reduction

- Let $\mathrm{BPP}(\alpha, \beta)$ denote the class of languages A such that there exists a polynomial time randomized algorithm M with the following properties:
  - If $x \in A$, $\Pr[M \text{ accepts}] \geq \beta$
  - If $x \notin A$, $\Pr[M \text{ accepts}] \leq \alpha$

  $(0 \leq \alpha < \beta \leq 1)$

  Note: probability is over the randomness of M, **not** over the inputs!
- When $\alpha = 0$ or $\beta = 1$ we call it “one sided error”
- It’s standard to define $\mathrm{BPP} = \mathrm{BPP}\left(\frac{1}{3}, \frac{2}{3}\right)$

Error Reduction

Claim: $\mathrm{BPP}\left(\frac{1}{2}, 1\right) = \mathrm{BPP}(0.000001,1)$

In words: if A is a language such that there's a polynomial algorithm M such that

- If $x \in A$, $\Pr[M \text{ accepts}] = 1$
- If $x \notin A$, $\Pr[M \text{ accepts}] \leq \frac{1}{2}$

Then there's a (different) algorithm M' such that

- If $x \in A$, $\Pr[M' \text{ accepts}] = 1$
- If $x \notin A$, $\Pr[M' \text{ accepts}] \leq 0.000001$

(The constant 0.000001 is of course arbitrary)

Error Reduction – One-sided case

Proof: Let ε be a small constant. Let M be an algorithm s.t.

- If $x \in A$, $\Pr[M \text{ accepts}] = 1$
- If $x \notin A$, $\Pr[M \text{ accepts}] \leq \frac{1}{2}$

Let M' be an algorithm that runs M $t = \log\left(\frac{1}{\varepsilon}\right)$ times, each time with fresh and independent random coins. If all runs accept, M' accepts. If at least one run rejects, M' rejects.

- If $x \in A$, $\Pr[M' \text{ accepts}] = 1$
- If $x \notin A$, $\Pr[M' \text{ accepts}] \leq \left(\frac{1}{2}\right)^t = \varepsilon$

Note: we can even pick $\varepsilon = 2^{-n}$. M' still runs in polynomial time and error probability is exponentially small. ■

Error Reduction – Two-sided Case

Claim: $\mathrm{BPP}\left(\frac{1}{3},\frac{2}{3}\right) = \mathrm{BPP}(0.000001,0.9999999)$

That is, we can also decrease the error probability of randomized algorithms with two-sided error.

Proof idea: Let M be an algorithm such that

- If $x \in A$, $\Pr[M \text{ accepts}] \geq \frac{2}{3}$
- If $x \notin A$, $\Pr[M \text{ accepts}] \leq \frac{1}{3}$

We'll run M for t many times and…

Take a majority vote.

Error Reduction – Two-sided Case

- Idea: if $x \in A$, we expect about $\frac{2}{3}$ of the runs to accept
- If $x \notin A$, we expect about $\frac{1}{3}$ of the runs to accept
- We accept if and only if more than half of the runs accepted
- We'll need mathematical some tools to analyze the error of this process

Large Deviation Inequalities

- Let $X_1, \ldots, X_m$ be i.i.d (independent, identically distributed) $\{0,1\}$ random variables such that $\Pr[X_1 = 0] = \frac{2}{3}$ and $\Pr[X_1 = 1] = \frac{1}{3}$
- (Think of X_i as denoting the output of the randomized algorithm in the case that the input x isn't in A. The case $x \in A$ is analogous so we'll only handle this case)
- M' is wrong if $\sum_{i=1}^{m} X_i \geq \frac{m}{2}$
- For every i, $\mathbb{E}[X_i] = 0 \cdot \frac{2}{3} + 1 \cdot \frac{1}{3} = \frac{1}{3}$
- $\mathbb{E}[\sum_{i=1}^{m} X_i] = \sum_{i=1}^{m} \mathbb{E}[X_i] = \frac{m}{3}$ (by linearity of expectation)
- On average, on m runs the algorithm accepts $\frac{m}{3}$ times
- What's the probability of the event $\sum_{i=1}^{m} X_i \geq \frac{m}{2}$?

Markov's Inequality

Claim: Let $X_1, \ldots, X_m$ be as before and $X = \sum_{i=1}^{m} X_i$ Then

$$\Pr[X \geq t] \leq \frac{\mathbb{E}[X]}{t}$$

Plugging $t = \frac{m}{2}$ and $\mathbb{E}[X] = \frac{m}{3}$, $\Pr[\sum_{i=1}^{m} X_i \geq t] \leq \frac{2}{3}$.

(not that great bound, but let's play along anyway)

Intuition: If $\Pr[X \geq t] > p$ then just summing over these values contributes more than $t \cdot p$ to the expectation.

Markov's Inequality

Claim: Let $X_1, \ldots, X_m$ be as before and $X = \sum_{i=1}^{m} X_i$ Then

$$\Pr[X \geq t] \leq \frac{\mathbb{E}[X]}{t}$$

Proof: let $\mathbf{1}_{X \geq t}$ denote the indicator random variable for the event "$X \geq t$":

$$\mathbf{1}_{X \geq t} = \begin{cases} 1 & \text{with probability } \Pr[X \geq t] \\ 0 & \text{otherwise} \end{cases}$$

Note that $t \cdot \mathbf{1}_{X \geq t} \leq X$. Take expectation from both sides to get

$$t \cdot \Pr[X \geq t] \leq \mathbb{E}[X]$$

Q: where did we use the fact that X_i's are independent?

$$\mathbb{E}[\mathbf{1}_{X \geq t}] = \mathbb{E}\left[\sum_{i=1}^{m} \mathbf{1}_{X_i \geq t}\right] = \sum_{i=1}^{m} \mathbb{E}[\mathbf{1}_{X_i \geq t}] = t \cdot P[X \geq t] + (m-t) \cdot 0$$

Chebyshev's Inequality

Markov: $\Pr[X \geq t] \leq \frac{\mathbb{E}[X]}{t}$

We didn't use the fact that X is a sum of n i.i.d. random vars.

Chebyshev's inequality: $\Pr[|X - \mathbb{E}[X]| \geq t] \leq \frac{\mathrm{var}(X)}{t^2}$

where $\mathrm{var}(X)$ is the **variance** of X:

$$\mathrm{var}(X) = \mathbb{E}[(X - \mathbb{E}[X])^2]$$

Variance measure how "far" X is from its expected value:

Small variance: X is almost always very close to $\mathbb{E}[X]$

High variance: X is "spread out"

Chebyshev's Inequality

Chebyshev's inequality: $\Pr[|X - \mathbb{E}[X]| \geq t] \leq \frac{\mathrm{var}(X)}{t^2}$

Proof:$|X - \mathbb{E}[X]| \geq t$ iff $|X - \mathbb{E}[X]|^2 \geq t^2$, and by Markov for $Y = |X - \mathbb{E}[X]|^2$

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

Chebyshev's Inequality

Chebyshev's inequality: $\Pr[|X - \mathbb{E}[X]| \geq t] \leq \frac{\mathrm{var}(X)}{t^2}$

Suppose $X = \sum_{i=1}^{m} X_i$ where $X_i = 1$ with probability p and 0 o/w.
Expected value: $\mathbb{E}[X] = \sum_{i=1}^{m} \mathbb{E}[X_i] = pm$

Let's calculate the variance of X.

$$\mathrm{var}(X) = \mathbb{E}[(X - pm)^2] = \mathbb{E}\left[\left(\sum_{i=1}^{m}(X_i - p)\right)^2\right]$$

$$= \mathbb{E}\left[\sum_{i=1}^{m}(X_i - p)^2 + \sum_{i \neq j}(X_i - p)(X_j - p)\right]$$

Chebyshev's Inequality

By linearity of expectation, this equals

$$\sum_{i=1}^{m} \mathbb{E}[(X_i - p)^2] + \sum_{i \neq j} \mathbb{E}\big[(X_i - p)\big(X_j - p\big)\big]$$

For every $i \neq j$, $X_i - p$ and $X_j - p$ are independent with expectation 0: so $\mathbb{E}\big[(X_i - p)\big(X_j - p\big)\big] = \mathbb{E}[X_i - p] \cdot \mathbb{E}\big[X_j - p\big] = 0$.

$$\mathbb{E}[(X_i - p)^2] = p \cdot (1-p)^2 + (1-p) \cdot p^2 = p(1-p)$$

In total, $\mathrm{var}(X) = m \cdot p(1-p) \leq pm$.

Chebyshev's Inequality

Chebyshev's inequality: $\Pr[|X - \mathbb{E}[X]| \geq t] \leq \frac{\mathrm{var}(X)}{t^2}$

Suppose $X = \sum_{i=1}^{m} X_i$ where $X_i = 1$ with probability p and 0 o/w. Then

$$\Pr[|X - pm| \geq t] \leq \frac{pm}{t^2}$$

if $t \gg \sqrt{m}$ this probability goes to 0 as $m \to \infty$.

Where did we use the fact that X is a sum of m i.i.d. random vars?

Chernoff Bound

So far we only used **pairwise independence**.

Chernoff's bound is a much stronger bound that requires stronger independence but gives much more:

Thm: Suppose $X = \sum_{i=1}^{m} X_i$ where X_i i.i.d. as before. Then

$$\Pr[|X - pm| \geq \epsilon m] \leq 2e^{-2\varepsilon^2 m}$$

The probability to significantly deviate from the expectation is exponentially small!

Proof of Chernoff Bound

We'll start with a "symmetric" case which makes some calculations easier.

$$X = X_1 + \cdots + X_m$$

where all X_i's are uniform over $\{-1,1\}$ and mutually independent. Note that $\mathbb{E}[X_i] = \mathbb{E}[X] = 0$.

Claim: for all $a > 0$, $\Pr[X > a] \leq e^{-\frac{a^2}{2m}}$.

Proof idea: in Chebyshev we squared Markov's inequality. Here we'll take the exponent.

Proof of Chernoff Bound

Claim: for all $a > 0$, $\Pr[X \geq a] \leq e^{-\frac{a^2}{2m}}$.

Proof: Let $\lambda > 0$ be a parameter to be fixed later.

$$\mathbb{E}\left[e^{\lambda X_i}\right] = \frac{e^{\lambda} + e^{-\lambda}}{2} \leq e^{\frac{\lambda^2}{2}}$$

(last inequality: by comparing Taylor expansions)

$$e^{\lambda X} = e^{\lambda(X_1 + \cdots + X_m)} = \prod_{i=1}^{m} e^{\lambda X_i}$$

$$\mathbb{E}\left[e^{\lambda X}\right] = \mathbb{E}\left[\prod_{i=1}^{m} e^{\lambda X_i}\right] = \prod_{i=1}^{m} \mathbb{E}\left[e^{\lambda X_i}\right] \leq e^{\frac{\lambda^2 m}{2}}$$

Proof of Chernoff Bound

Claim: for all $a > 0$, $\Pr[X \geq a] \leq e^{-\frac{a^2}{2m}}$.

Now use Markov:

$$\Pr[X \geq a] = \Pr\left[e^{\lambda X} \geq e^{\lambda a}\right] \leq \frac{\mathbb{E}\left[e^{\lambda X}\right]}{e^{\lambda a}} \leq e^{\frac{\lambda^2 m}{2} - \lambda a}$$

Set $\lambda = \frac{a}{m}$ to minimize the RHS to get the bound.

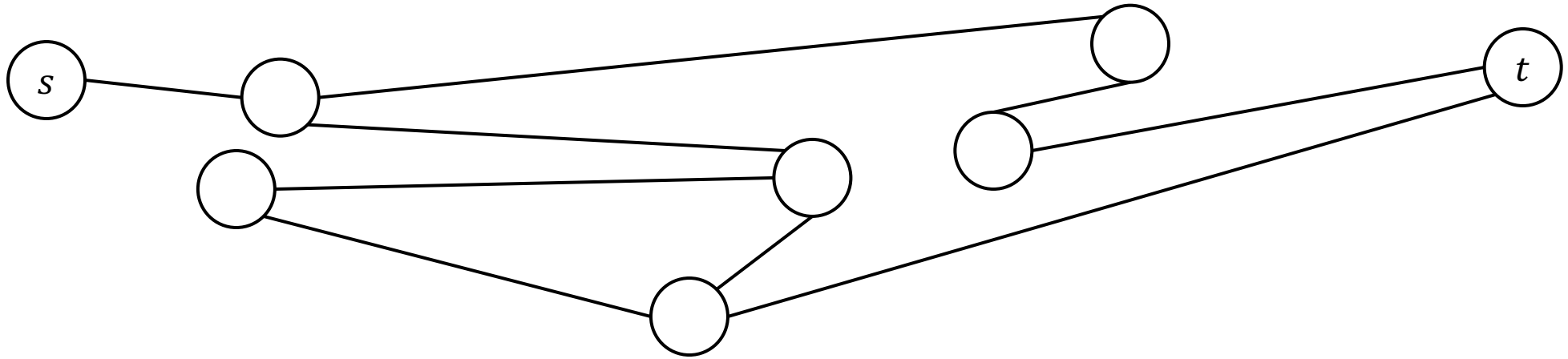
By symmetry:

$$\Pr[|X| \geq a] \leq 2e^{-\frac{a^2}{2m}}$$

You'll prove the more general case in HW (similar proof).

Undirected Connectivity

Problem: given an **undirected graph** G and two nodes s, t, decide if there's a path from s to t in G.



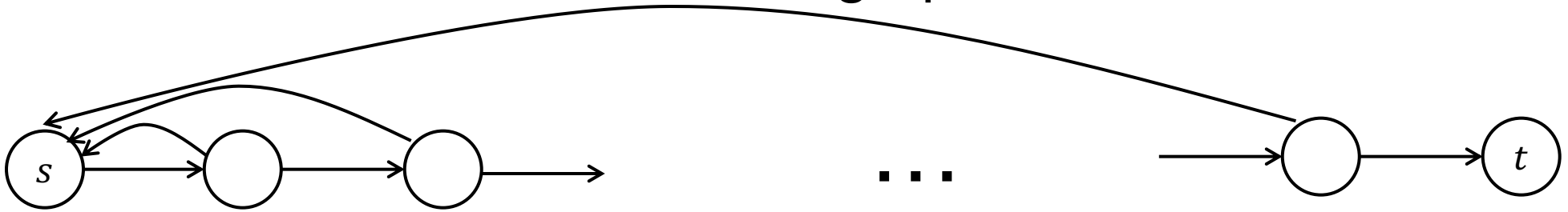
Looks like a very **easy** problem (just run BFS/DFS)

Undirected Connectivity

- DFS/BFS requires saving a copy of the entire graph in the memory
- We want to solve this problem with very little additional memory
- Recall complexity theory course: we want a randomized algorithm that runs in space $O(\log n)$
- Informally: “a constant number of variables”

Undirected Connectivity

- How to travel from s to t without remembering anything?
- Idea: take a **random walk** from s to t: start with s and at each step pick a **random neighbor** in the graph
- Hope: that in few steps we'll reach t with high probability
- This doesn't work for **directed** graphs:



- We'll show that for **undirected** graphs it works pretty well

Undirected Connectivity

Input: An undirected graph G with n vertices, and two vertices s, t

Output: "yes" if there's a path from s to t, "no" otherwise

1. Set $u = s$
2. Do n^5 times:
    1. Pick a random neighbor v of u
    2. If $v = t$ output "yes" and halt
    3. Set $u = v$
3. Output "no"

הערה:
אפשר להוכיח בעזרת
שא סיבוכיות טובה יותר
$O(n^3)$-מ

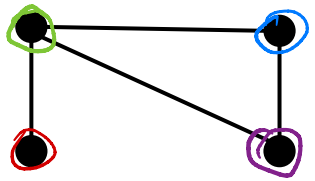
Random Walks on Graphs

- This process is called a **random walk** on G
- Easy: if there's no path between s and t, this algorithm will never reach t and will always output "no"
- Also easy: this algorithm runs in logarithmic space (need to store u, v and count to n^5)
- Need to show: if t reachable from s, this process reaches t with some noticeable probability
- We need tools to analyze the distribution on vertices obtained by making i steps in this walk

Random Walks on Graphs

- Simplifying assumption: G is a d-regular graph (can assume wlog by adding self loops, doesn't affect reachability of t from s)
- Can also assume G is connected (otherwise focus on the connected component of s and t)
- We'll show that after enough steps, the distribution of this process becomes "close to uniform"
- $\Rightarrow$ Probability to reach t is $\geq \frac{1}{2n}$ (in uniform dist. it's $\frac{1}{n}$)
- By repeating the algorithm n^2 times, can make it ≥ 0.999

Adjacency Matrix



$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}$$

Let A be the normalized adjacency matrix of G: $A_{u,v} = \frac{1}{d} \cdot (\#\text{ edges between } u \text{ and } v)$

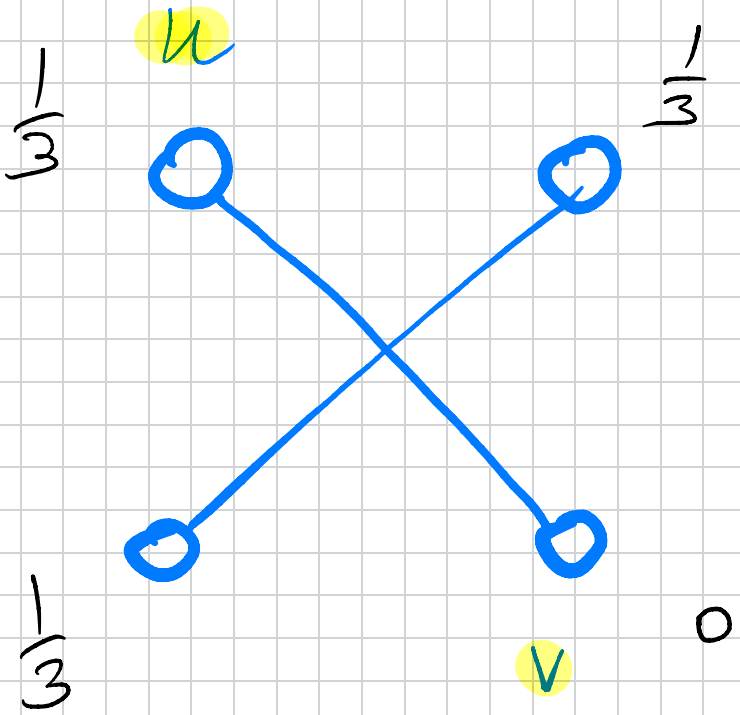
(we allow multiple edges and self loops)

Claim: Let $q = (q_1 \ldots, q_n) \in \mathbb{R}^n$ be an distribution over the n vertices. Then Aq is the distribution of a "one step random walk"

1. Picking a vertex according to q
2. Picking a random neighbor

Proof: For every v, $A_{u,v} =$ the probability of going from u to v.

$$(Aq)_v = \sum_u A_{u,v} \cdot q_u$$


u
1/3
1/3
1/3
0
v

Random Walk Operator

Claim: Let $p_0 \in \mathbb{R}^n$ be an distribution over the n vertices. Then $A^i p_0$ is the distribution p_i obtained by:

1. Picking a vertex according to p_0
2. Doing a random walk from that vertex for i steps.

Proof: induction on i.

In our case, p_0 is the vector which is 1 in s-th coord., 0 elsewhere

$p_0 = [0 \ldots 0 0 \underset{s}{1} 0 \ldots 0]$

How to compute $A^i p_0$?

There are vectors for which it is very easy to compute the action of A on them…

Eigenvalues and Eigenvectors of A

Note: A is symmetric, sum of every row is 1.

Thm: This implies that A has n **real** eigenvalues

$$-1 \leq \mu_n \leq \mu_{n-1} \leq \cdots \leq \mu_1 = 1$$

And the corresponding eigenvectors $v_n, \ldots, v_1 = \frac{1}{\sqrt{n}}\mathbf{1}$ are **orthonormal**

For technical reasons: we'll work with the "lazy random walk" matrix $B = \frac{1}{2}I + \frac{1}{2}A$ which has the same eigenvectors with eigenvalues

$$0 \leq \lambda_n \leq \lambda_{n-1} \leq \cdots \leq \lambda_1 = 1 \qquad \lambda_i = \frac{1}{2} + \frac{1}{2}\mu_i$$

"With probability $\frac{1}{2}$ stay where you are, with probability $\frac{1}{2}$ make one step in the graph"

Spectral Graph Theory

The eigenvalues of A contain a lot of information on G.

For example: G bipartite iff $\mu_n = -1$, G connected iff $\mu_2 < 1$

We'll only need two facts:

Fact 1: $\lambda_1 = \mu_1 = 1$ and $v_1 = \frac{1}{\sqrt{n}} \cdot (1,1,\ldots,1)$ (because all rows sum to 1)

The vector v_1 corresponds to the uniform distribution

Fact 2: If G is connected, $\lambda_2 \leq 1 - \frac{1}{4n^4}$

Intuition for what's going on

Write $p_0 = \sum_i \alpha_i v_i$ as a linear combination of the eigenvectors

It'll now be easy to compute $B^k p_0$ by linearity since v_i are e.v's

All components except the one corresponding to v_1 will be small and tend to 0 quickly as k grows

We'll be left with "uniform distribution + small error"

A bit more formally

$p_0 = \sum_i \alpha_i v_i$

$\langle v_i, p_0 \rangle = \langle v_i, \sum_j \alpha_j v_i \rangle = \sum_j \alpha_j \langle v_i, v_j \rangle = \alpha_i$

- Write $p_0 = \sum_i \alpha_i v_i$ as a linear combination of the eigenvectors
- By orthonormality, $\alpha_i = \langle p_0, v_i \rangle$
  - $\alpha_1 = \langle p_0, v_1 \rangle = \left\langle p_0, \frac{1}{\sqrt{n}}(1, \ldots, 1) \right\rangle = \frac{1}{\sqrt{n}}$ נורמליזציה
  - $|\alpha_i| = |\langle p_0, v_i \rangle| \leq \|p_0\| \cdot \|v_i\| = 1$ for all i
- $B^k p_0 = \sum_i \alpha_i B^k v_i = \sum_i \alpha_i \lambda_i^k v_i = \frac{1}{\sqrt{n}} \cdot \frac{1}{\sqrt{n}} \cdot (1, \ldots, 1) + \sum_{i=2}^{n} \alpha_i \lambda_i^k v_i$
- As if by magic, the first term is the uniform distribution!
- We'll show that the second ~~them~~ term is a vector of small norm

$B^k p_0 = \sum_{i=1}^{n} \alpha_i B^k v_i = \sum_{i=1}^{n} \alpha_i \lambda_i^k v_i = \alpha_1 \lambda_1^k v_1 + \sum_{i=2}^{n} \alpha_i \lambda_i^k v_i = \frac{1}{\sqrt{n}} \cdot 1^k \cdot \frac{1}{\sqrt{n}}(1, \ldots, 1) + \sum_{i=2}^{n} \alpha_i \lambda_i^k v_i$

Calculating the error

$$\left\| B^k p_0 - \frac{1}{n}(1, \ldots, 1) \right\|^2 = \left\| \sum_{i=2}^{n} \alpha_i \lambda_i^k v_i \right\|^2$$

By orthonormality of v_i's, the right hand side equals

$$\left\langle \sum_{i=2}^{n} \alpha_i \lambda_i^k v_i \, , \sum_{i=2}^{n} \alpha_i \lambda_i^k v_i \right\rangle = \sum_{i=2}^{n} \alpha_i^2 \lambda_i^{2k}$$

(every cross term is zero). Since $\lambda_2 \geq \cdots \geq \lambda_n \geq 0$, this is at most

$$\geq \lambda_2^{2k} \sum_{i=2}^{n} \alpha_i^2 = \lambda_2^{2k} \|p_0\|^2 = \lambda_2^{2k}$$

Bounding the error

$\leq n\left(1-\frac{1}{4n^4}\right)^{4n^5} \leq ne^{-n}$

We showed: $\left\|B^k p_0 - \frac{1}{n}(1,\ldots,1)\right\|^2 \leq \lambda_2^{2k}$.

You'll prove in HW: $\lambda_2 \leq 1 - \frac{1}{4n^4}$ (this isn't the tightest bound)

If $k = 4n^5$, $\left\|B^k p_0 - \frac{1}{n}(1,\ldots,1)\right\|^2 \leq e^{-n}$

So the probability to reach t after $4n^5$ steps is at least $\frac{1}{n} - e^{-\frac{n}{2}} \geq \frac{1}{2n}$.

מסקנה: לאחר $4n^5$ צעדים, מקבלים התפלגות שאינה רחוקה מהתפלגות אחידה.

Recap

We wanted to prove: if s and t are connected, then a random walk starting from s reaches t after $O(n^5)$ steps with decent probability.

We proved: in a connected graph, starting from any distribution, after polynomially many steps we're close to the **uniform** distribution

We showed this by analyzing the eigenvalues of the adjacency matrix. Such techniques are called **spectral** techniques.

Application: 2SAT

In the 2SAT problem we're given a 2CNF formula and want to decide if it's satisfiable

$$(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_3) \wedge \cdots \wedge (x_8 \vee x_n)$$

3SAT is NP-complete and thus likely not efficiently solvable.

There's a deterministic (slightly complicated) algorithm for 2SAT.

We'll see a simple randomized algorithm.

Randomized 2SAT Algorithm

Input: a 2CNF formula φ with n variables

Output: "yes" if φ is satisfiable, "no" otherwise

1. Pick a random assignment β
2. Do n^5 times:
   1. If φ is satisfied by β output "yes" and halt.
   2. Otherwise, pick an arbitrary clause $(\ell_1 \vee \ell_2)$ unsatisfied by β
   3. Pick **randomly** one of the variables in the clause and flip its value
3. Output "no"

Randomized 2SAT Algorithm

Claim 1: the algorithm runs in polynomial time (obvious)

Claim 2: if φ is unsatisfiable, the algorithm outputs “no” (also obvious)

Left to prove: if φ is satisfiable, the algorithm outputs “yes” with high probability.

We’ll show this follows from the analysis of the previous algorithm.

2SAT Algorithm – Analysis

Suppose φ is satisfiable and let α be a satisfying assignment (if there's more than one, pick one arbitrarily). α satisfies all clauses in φ.

If β doesn't satisfy a clause, there's at least one variable in that clause for which β and α give different values.

Define $\mathrm{dist}(\beta, \alpha) = |\{i : \alpha(x_i) \neq \beta(x_i)\}|$.

On each update step of β we:

- Decrease the distance by 1 with probability $\geq \frac{1}{2}$
- Increase the distance by 1 with probability $\leq \frac{1}{2}$

$\alpha = (0, 1, 0, 0, 1, 0)$

$\beta = (0, 1, 0, 0, 0, 1)$

$dist(\alpha, \beta) = 2$

Hamming Distance

דוגמה:

$(X_1 \cup X_2)$

$\alpha = 1,0$
$0,1$
$1,1$

$\beta = 0,0$

$\beta = 1$ ⟸ בשלב הבא
0

$\alpha = 1$
המחלק קונה
קלן
$\alpha = 0$ { שקיבל $\frac{1}{2}$ / שלא קיבל $\frac{1}{2}$

קלן ב-1 המחלק $\frac{2}{3}$ בהסתברות ולכן

צלל ב-1 $\frac{1}{3}$

2SAT Algorithm – Analysis

Consider the line graph:

Interpret being at vertex i as having distance i from α.

We don't know in which vertex we start, but at each step we make

- one step left with probability $\geq \frac{1}{2}$
- one step right with probability $\leq \frac{1}{2}$.

Suppose both probabilities are exactly $\frac{1}{2}$ (this can only slow us down)

2SAT Algorithm – Analysis

After at most $O(n^5)$ steps we'll reach the vertex 0 with high probability

Being at distance 0 from α means $\beta = \alpha$ and satisfying

(Of course there may be other satisfying assignments and we may find another one along the way. This can only help us)

Why doesn't this work for 3SAT?

Isolation Lemma

- Let's play fantasy soccer!
- The FIFA World Cup is taking place
- There are ~ 100 players
- You can select a team of 11 players
- By the end of the tournament, each player gets a score between 0 and 1000 (maybe number of goals scored or saves or whatever…)
- Team's score = **sum** of individual players' scores
- Highest scoring team wins a prize

Fantasy Soccer

- You get 10^7 entries, all with distinct teams
- Note: # of possible teams is $\binom{100}{11} \approx 10^{14}$
- What if there are too many teams with highest score?
- Team's max score is between 0 and $11{,}000$
- Many more teams than scores, so there will be many teams with the same score
- Will there be $\frac{10^7}{11000} \approx 900$ first prize winners?
- Theorem: with high probability, **unique** first prize winner!

Isolation Lemma

For a weight function $w: \{1,2,\dots,n\} \to \{1,2,\dots,N\}$, the weight of a set $S \subseteq \{1,2,\dots,n\}$ is defined as $w(S) = \sum_{i \in S} w(i)$.

Lemma: Let $\mathcal{F}$ be a family of subsets of $\{1,2,\dots,n\}$. Suppose each $x \in \{1,\dots,n\}$ is assigned an independent, uniform random weight from $\{1,2,\dots,N\}$. Then, with probability **at least** $1 - \frac{n}{N}$, there's a **unique** set in $\mathcal{F}$ that has minimal weight among all sets in $\mathcal{F}$.

Isolation Lemma

Lemma: Let $\mathcal{F}$ be a family of subsets of $\{1,2,\dots,n\}$. Suppose each $x \in \{1,\dots,n\}$ is assigned an independent, uniform random weight from $\{1,2,\dots,N\}$. Then, with probability **at least** $1-\frac{n}{N}$, there's a **unique** set in $\mathcal{F}$ that has minimal weight among all sets in $\mathcal{F}$.

Remarks:

1. Only meaningful when $N > n$
2. Same holds with **maximum** weight
3. No dependence on the size or structure of $\mathcal{F}$!

Proof of Isolation Lemma

Proof: for any $x \in \{1,2,\dots,n\}$, define

$$\alpha(x) = \min_{S \in \mathcal{F}, x \notin S} w(S) - \min_{S \in \mathcal{F}, x \in S} w(S \setminus \{x\})$$

If there are two sets A, B of minimum weight, then for $x \in A \setminus B$:

$$\alpha(x) = w(B) - \big(w(A) - w(x)\big) = w(x)$$

On the other hand, $\alpha(x)$ is independent of $w(x)$, and $w(x)$ is chosen uniformly from $\{1,2,\dots,N\}$. So $\alpha(x) = w(x)$ with probability at most $\frac{1}{N}$.

The probability that for **some** x, $w(x) = \alpha(x)$ is $\leq \frac{n}{N}$.

Union Bound

דוגמה

$1 \rightarrow 0$

$2 \rightarrow 5$

$3 \rightarrow 10$

$F = \{ \{1\}\ \{2\}\ \{3\}\ \{1,2\}\ \{1,3\}\ \{2,3\}\ \{1,2,3\}\ \{\phi\} \}$

$n = \{1, 2, 3\}$

$N = [0, 10]$

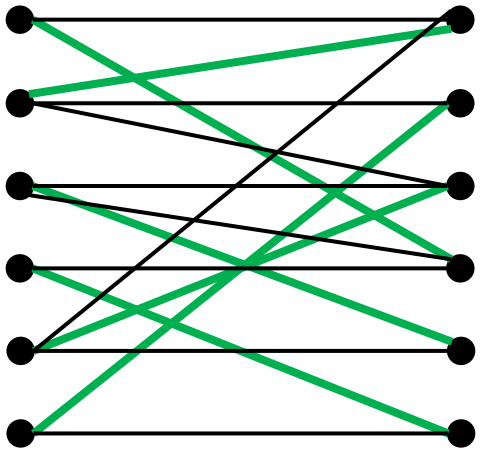
נניח $X = 1$ (0 בין 3)

$B = \{2\}$ 5

$A = \{1\}$ 0

Isolation Lemma and Perfect Matching

- Given G, assign each edge (i, j) a weight $w_{i,j}$ uniformly at random from $\{1,2, \ldots, 10m\}$ where $m = |E|$
- If there exists a perfect matching, then with probability at least $\frac{9}{10}$, there's a unique perfect matching with minimal weight
- How to find it?



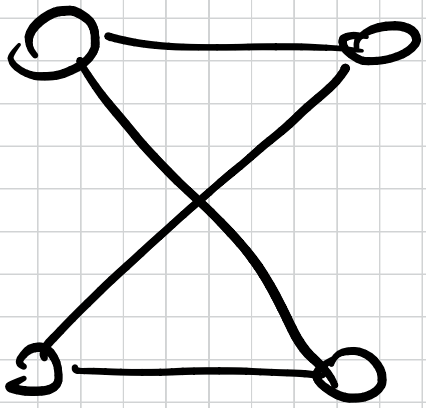
F - משפחת כל הקבוצות
של קשתות שמהוות שידוך
מושלם

$w=n$
$N=10m$
$1-\frac{m}{10m}=\frac{9}{10}$

הוכחה:
לפחות הפתרון שמתקבל ב- G הוא
יחיד מ"ב.

Isolation Lemma and Perfect Matching

- Let's start be (re-)solving the decision version
- Let X be the symbolic biadjacency matrix of G
  - In cell (i,j): $x_{i,j}$ if (i,j) is an edge, 0 otherwise
- Replace $x_{i,j}$ by $2^{w_{i,j}}$ where $w_{i,j}$ is the weight of edge (i,j)
- We'll show that assuming there's a unique min weight perfect matching, $\det X$ is non-zero
- For a permutation σ, $\prod_{i=1}^{n} x_{i,\sigma(i)} = 2^{w(\sigma)}$ if σ is a perfect matching in G, and 0 otherwise



$$\begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix}$$

דוגמא:

מקצים משקולות אקראיים w_{ij}

לכל קשת x_{ij} :

$$x_{ij} \leftarrow 2^{w_{ij}}$$

נחשב דטרמיננטה:

$$\left| \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} \right| = 2^{w_{11}\cdot w_{22} - w_{12}\cdot w_{21}}$$

או באופן כללי:

$$\det X = \sum_{\sigma:} (-1)^{sign(\sigma)} \cdot 2^{w(\sigma)}$$

σ שידוך מושלם

מסקנה! אם קיים שידוך מושלם יחיד משקל מינימלי $\Leftarrow \det X \neq 0$.

מה המשקל המינימלי של השידוך המושלם?

החזקה הכי גדולה k של 2^k שמחלקת את הסכום

משקל מינימלי

שים לב! הסכום הנ"ל תמיד אי זוגי. נקבל 2^0 ועוד סכומים זוגיים. מורכב מ- $\pm 2^{k} \pm$(חזקות גדולות יותר של 2)

Isolation Lemma and Perfect Matching

- For a permutation σ, $\prod_{i=1}^{n} x_{i,\sigma(i)} = 2^{w(\sigma)}$ if σ is a perfect matching in G, and 0 otherwise
- Suppose σ_0 is the unique min weight perfect matching

$$\det X = \sum_{\text{perfect matchings } \sigma} (-1)^{\operatorname{sgn}\sigma} 2^{w(\sigma)}$$

$$= \pm 2^{w(\sigma_0)} + \text{higher powers of 2 (or zero)}$$

- $\Rightarrow$ if there's a perfect matching and min weight is unique, $\det(X) \neq 0$ and $2^{w(\sigma_0)}$ largest power of 2 dividing $\det X$

On to finding a perfect matching

- For each edge e:
  - Decrease the weight of e by 1
    - (let's say we picked the weight to be ≥ 2 in the first place – that's fine)
  - Compute the determinant again
  - See if minimum weight decreased – if yes, the edge participates in a min weight perfect matching
- This can be done "in parallel" for all edges

Parallel Matching

- In fact, all the algorithm above can be “parallelized”
- Assume you have polynomial number of processors that can operate in parallel
- Determinant can be computed in this model in time $O(\log^2 n)$ (we’ll prove it later in class)
- For finding the matching, we need to assign weights and compute n^2 determinants in parallel
- We get a parallel randomized algorithm that finds a perfect matching (max flow algorithm are highly sequential)

Other uses of the Isolation Lemma

- The isolation lemma can be also used to show **hardness** results
- For example: we know it's NP-hard to find the maximum clique in a graph
- Does it become easier if the clique is unique?
- We'll show that probably not by giving a **randomized reduction** from CLIQUE to UNIQUE-CLIQUE

NP hardness refresher

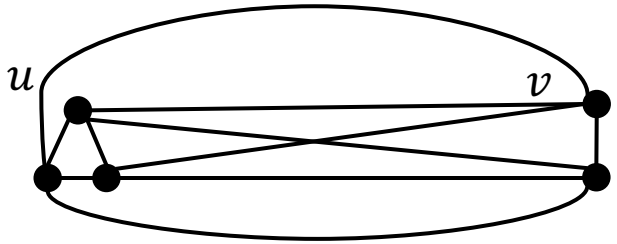
- NP = class of languages that can be verified in polynomial time given a “witness” / “certificate”
- CLIQUE = $\{\langle G, k\rangle : G \text{ has a clique of size } k\}$ is **NP-complete**
- CLIQUE is in NP and for a every other language in NP there’s a **reduction** to CLIQUE:
  - Reduction from A to B: a polynomial time computable function such that $w \in A$ iff $f(w) \in B$
- UNIQUE-CLIQUE: is there **exactly one** clique of size k?
- We might expect that it’s easier to find a unique clique

UNIQUE-CLIQUE is Hard

- We'll show a **randomized reduction** from CLIQUE to UNIQUE-CLIQUE
  - A function computable in randomized polynomial time such that $f(\langle G, k\rangle) = \langle G', k'\rangle$ and
    - If $\langle G, k\rangle \notin \text{CLIQUE}$ then $\langle G', k'\rangle \notin \text{UNIQUE}-\text{CLIQUE}$
    - If $\langle G, k\rangle \in \text{CLIQUE}$ then $\langle G', k'\rangle \in \text{UNIQUE}-\text{CLIQUE}$ with probability $\geq \frac{1}{4nk}$
- Start by picking a random weight $w(v) \in \{1,2, \ldots, 2n\}$ for each vertex v in G $(n = |V|)$
- With probability at least $\frac{1}{2}$, max weight clique of size k is **unique**

Reduction to UNIQUE-CLIQUE

- Construct G' as follows: each vertex v is replaced by a clique on $2nk + w(v)$ vertices. For each edge (u, v) in G, each copy of u is connected to each copy of v



- Pick random $r \in [1,2, \ldots, 2nk]$ and let $k' = 2nk^2 + r$
- Output $\langle G', k' \rangle$

Analysis of the reduction

- If $\langle G, k \rangle \notin \text{CLIQUE}$, the size of the largest clique in G' is at most $(k-1) \cdot (2nk + 2n) < 2nk^2$ so $\langle G', k' \rangle \notin \text{UNIQUE}-\text{CLIQUE}$ (no matter which r is picked)
- If $\langle G, k \rangle \in \text{CLIQUE}$, with probability at least $\frac{1}{2}$, there's a **unique** clique of size k with max weight under w, corresponding to a clique of size $2nk^2 + w(C)$ in G'.
- $1 \leq w(C) \leq 2nk$, and $r = w(C)$ with probability $\frac{1}{2nk}$.
- Hence, with probability $\geq \frac{1}{4nk}$ there's a unique clique of size k'

Epilogue

- We’ve seen many examples for randomized algorithm
- Where to random bits come from?
- Deterministic algorithms have an advantage: they don’t need random bits
- Can we always convert randomized algorithm to deterministic algorithms?
- In many cases we know how to do this, but not in all

Epilogue

- Derandomization and Pseudorandomness – big and fascinating topics in complexity theory
- Derandomization is related to proving hardness results
- For example, there's a "theorem" which states that a deterministic algorithm for PIT implies that "P≠NP"
- Proving that one thing is easy implies that other things are hard!
- Next up: algebraic algorithms