CS 761: Randomized Algorithms **University of Waterloo: Winter 2025**

Lecture 14

March 6, 2025

Instructor: Sepehr Assadi

Topics of this Lecture

1 Proof of Schwartz-Zippel Lemma

Recall the Schwartz-Zippel Lemma [Zip79], [Sch80] from Lecture 13 which states:

Lemma 1. *Let $P(x_1, \ldots, x_n)$ be a polynomial with degree d. If we sample $(x_1, \ldots x_n)$ randomly from S^n,*

$$\Pr\left(P(x_1, \ldots, x_n) = 0\right) \leqslant \frac{d}{|S|}.$$

Proof. We prove this Lemma by induction on n.

Base Case: when $n = 1$, P is a univariate polynomial in x_1 with at most d roots and thus

$$\Pr\left(P(x_1) = 0\right) = \frac{d}{|S|}.$$

For the inductive step, we pick x_1 and factoring it out, we get,

$$P(x_1, \ldots, x_n) = Q_1 \cdot x_1^k + Q_2 \cdot x_1^{k-1} + \ldots + Q_k \cdot x_1^0 \tag{1}$$

Note that Q_1 is a non-zero polynomial with degree at most $d - k$. If we pick $x_2, \ldots, x_n$ randomly, then

$$\Pr\left(Q_1(x_2, \ldots, x_n) = 0\right) \leqslant \frac{d-k}{|S|}$$

by the induction hypothesis.

Fix a choice $r_2, \ldots, r_n$ of $x_2, \ldots, x_n$ such that $Q_1(r_2, \ldots, r_n)$ is non-zero. We define another polynomial

$$P'(x_1) = x_1^k \cdot Q_1(r_2, \ldots, r_n) + R(x_1),$$

where $R(x_1)$ represents the polynomial associated with $x^{k-1}, \ldots, x^0$ in Eq (1). It is clear that $P'(x_1)$ has degree of k. Thus, the probability of $P'(x_1) = 0$, given $Q(r_2, \ldots r_n) \neq 0$, is less than $\frac{k}{|S|}$. Putting these together:

$$\begin{aligned}\Pr\left(P(x_1, \ldots, x_n) = 0\right) &\leqslant \Pr\left(Q(x_2, \ldots, x_n) = 0\right) + \Pr\left(P'(x_1) = 0 \mid Q(x_2, \ldots, x_n) \neq 0\right) \\ &\leqslant \frac{d-k}{|S|} + \frac{k}{|S|} = \frac{d}{|S|}.\end{aligned}$$

□

2 A Parallel Algorithm for Bipartite Matching

In the last lecture, we saw an algorithm for deciding if a bipartite graph $G = (L, R, E)$ has a perfect matching or not, using algebraic techniques. We now design an algorithm for *finding* a perfect matching also when it exists.

The idea of the algorithm is simply to eliminate edges and perform polynomial testing to verify the existence of a perfect matching. The algorithm would run as follows:

1. Run the algorithm of last lecture to decide if G has a perfect matching or not, and if not, terminate. Otherwise, continue.
2. Pick an edge $e = (u, v)$ from G and remove it.
3. Run the last lecture algorithm again and if it says $G \setminus e$ has no perfect matching, we know e belongs to every perfect matching of G and we output it; otherwise, we remove it since there is another perfect matching even without this edge.
4. We then go back Step (2) on the graph $G \setminus e$.

Notice that the algorithm is sequential and not efficient. We want to **parallelize** this process to find perfect matchings more efficiently. We will not fully define the parallel model we work with (known as PRAM) and only mention the following result about parallel computation which we need:

Proposition 2. *Matrix multiplication and determinant of matrices with* $\text{poly}(n)$*-bit integers can be computed in parallel using* $\text{poly}(n)$ *processors and* $\text{poly}\log(n)$ *time.*

Our goal now is to describe a parallel algorithm, due to [MVV87], which can find the perfect matching in polylog(n) time with poly(n) processors. The challenge with parallelizing the above algorithm is that the perfect matching of G might not be unique. *If* we do have a **unique perfect matching**, then the parallelization of the reduction process will be trivial by simply running step (2) simultaneously for all edges in parallel instead of sequentially.

The fix to this is the so-called isolation lemma, another nice trick with randomization.

2.1 Isolation Lemma

The following is the so-called isolation lemma of [MVV87].

Lemma 3. *Let $\mathcal{F}$ be any collection of subsets of the universe $[n]$. Suppose we pick weights $w_1, \ldots, w_n$ for the elements of $[n]$ randomly and independently from $[N]$. Define weight of a set $S \subseteq [n]$ as $w(S) := \sum_{i \in S} w_i$. Then, the minimum weight set $S \in \mathcal{F}$ is unique with probability at least $1 - n/N$.*

Before getting to the proof, note that there is no restriction on the size of $\mathcal{F}$ in this result, and as long as we set N, say, n^2, we obtain the uniqueness of the minimum weight set with high probability, even when

$|\mathcal{F}|$ can be as large as $2^{\Omega(n)}$; note that in this case, since weight of any set is an integer from 1 to n^3, by pigeonhole principle, there are $|\mathcal{F}|/n^3 = 2^{\Omega(n)}$ sets all with the same weight and yet the minimum weight set is unique!

Proof. Fix any element e and define the sets $\mathcal{A}_e \subseteq \mathcal{F}$ as the ones that contain e, and $\mathcal{B}_e \subseteq \mathcal{F}$ as the ones that does not contain e. Define the following two quantities:

$$a_e := \min_{A \in \mathcal{A}_e} w(A \setminus e) \qquad \text{and} \qquad b_e := \min_{B \in \mathcal{B}_e} w(B).$$

Now, consider the following cases:

- **Case 1.** $a_e < b_e + w_e$ this means e does not belong to any minimum weight set;
- **Case 2.** $a_e > b_e + w_e$: this means e belongs to every minimum weight set;
- **Case 3.** $a_e = b_e + w_e$: this means e belongs to some but not all minimum weight sets (*ambiguous* case).

If no element $e \in [n]$ belongs to the ambiguous case, we have that the minimum weight set in $\mathcal{F}$ is unique. Thus, when we pick weights randomly, by union bound,

$$\Pr\left(\text{minimum weight set is not unique}\right) \leqslant \sum_{e \in [n]} \Pr\left(w_e = b_e - a_e\right).$$

On the other hand, note that both quantities a_e and b_e are functions of all weights except for w_e and thus their values is chosen independent of w_e in the sampling process. Since w_e is chosen from $[N]$, the chance it is equal to $b_e - a_e$ is at most $1/N$ (the reason for 'at most' and not 'equality' is that it is possible for $b_e - a_e$ to be outside $[N]$ even). Plugging in this bound above implies that

$$\Pr\left(\text{minimum weight set is not unique}\right) \leqslant \sum_{e \in [n]} \Pr\left(w_e = b_e - a_e\right) \leqslant n \cdot \frac{1}{N},$$

concluding the proof. □

2.2 Isolation Lemma for Parallel Matching

We now go back to our goal of designing a parallel matching algorithm. Let $G = (L, R, E)$ be any bipartite graphs with $|L| = |R| = n$ and suppose we pick weights w_e for $e \in E$. For any perfect matching M in G (assuming it exists), we define $w(M)$ as the weight of this perfect matching. By picking the weights w_e for $e \in E$ uniformly and independently from $[n^3]$ (since $|E| \leqslant n^2$), we can apply the isolation lemma (Lemma 3) to ensure that if G has a perfect matching, then with high probability, it will also have a minimum weight perfect matching. Conditioned on this, we show that we can easily run our sequential algorithm from earlier in parallel: by removing an edge e and checking if the minimum weight perfect matching of $G \setminus e$ change weight or not; and do this in *parallel* for all edges. We thus only need to have an algorithm that computes the weight of a minimum weight perfect matching in G. We do so using algebraic techniques.

Consider the $n \times n$ matrix $A = A(G)$ where rows corresponds to vertices in L, columns corresponds to vertices in R, and the value of $A_{u,v}$ for $u \in L$ and $v \in R$ is 0 if there is no (u, v) edge, and otherwise it is $2^{w_{u,v}}$, where $w_{u,v}$ is the weight of the edge (u, v). Consider the determinant of A, denoted by $\text{Det}(A)$, and recall from the last lecture that it is:

$$\text{Det}(A) = \sum_{M\text{: perfect matching}} sign(M) \prod_{(u,v) \in M} 2^{w_{u,v}} = \sum_{M\text{: perfect matching}} sign(M) \cdot 2^{w(M)};$$

(recall that $sign(M)$ denotes the sign of the permutation corresponding to the matching and for our purpose, it only matters that it is either $+1$ or -1).

Assuming G has a perfect matching, by writing $\mathrm{Det}(A)$ in the binary representation, we get that the right-most one-bit corresponds to the weight of the minimum weight matching. Thus, we can recover the weight of the minimum weight perfect matching from $\mathrm{Det}(A)$ in this case.

This way, we can design our final parallel algorithm for the bipartite perfect matching problem:

Algorithm 1.

1. Pick a random weight for each edge $e \in G$ independently and uniformly from $[n^3]$.
2. Compute $\mathrm{Det}(A)$ using Proposition 2; if it is zero, declare G has no perfect matching; else, let w^* be the weight corresponding to the right-most one-bit of its binary representation.
3. For each edge $e \in E$ in parallel:
    (a) Compute $\mathrm{Det}(A - \mathbf{1}_e \cdot 2^{w_e})$ using Proposition 2, namely, the determinant of the matrix whose entry corresponding to the edge e is zero out; if its right-most one-bit is w^* output e, otherwise ignore the edge e.

By Isolation Lemma (Lemma 3), the minimum weight perfect matching is unique in G with high probability, and by our preceeding discussion, this means the algorithm exactly outputs edges of this perfect matching. By Proposition 2, each determinant computation can be done in $\mathrm{poly}\log(n)$ time and since we handle all edges in parallel, we only need $\mathrm{poly}\log(n)$ time in total also while still using $\mathrm{poly}(n)$ processors given all numbers we use have $\mathrm{poly}(n)$-bit complexity.

3 Arthur-Merlin Communication

We now see yet another surprising application of the polynomial method, this time to communication complexity, using a result of [AW08]. Consider the following communication setting: Alice has a set $S \subseteq [n]$, and Bob has $T \subseteq [n]$. Alice will send one message to Bob. We also have a third communicator, Merlin, who sees both Alice's and Bob's inputs.

Rules:

- Merlin is only allowed to communicate with Bob, and Alice can't see Merlin's message to Bob.
- Merlin is untrustworthy; he may choose to send incorrect information to Bob.
- Merlin goes first. Then Alice sends a message to Bob. Finally, will attempt to answer the question whether $S \cap T = \emptyset$.

We want a protocol Π for this problem such that:

1. if Merlin is honest, Π is always correct.
2. if Merlin is not honest, $\Pr(\Pi \text{ is wrong}) \leqslant \frac{1}{poly(n)}$ (Π is allowed to output Merlin is lying instead of outputting a solution to the problem).

Ideally, we want Merlin to be easy enough to detect lies, but helpful enough to solve the problem. We can force Merlin to send a desired number of elements to Bob, such as 10 numbers (if Merlin doesn't at least follow *some* protocol, it will be obvious to Bob that he is lying). We have the following result on this game:

Theorem 4 ([AW08])**.** *There exists a protocol Π for this problem using $O(\sqrt{n}\log(n))$ bits of communication.*

Proof. We will use polynomial methods. Let consider the input as a grid with size $[\sqrt{n}] \times [\sqrt{n}]$ and the output is $\in \{0,1\}$. Then, we interpret Alice's set S as a function $A(x,y) \to \{0,1\}$, and Bob's set T as a function $B(x,y) \to \{0,1\}$. Define $A(x,y)$ as $A(x,y) = 1 \Leftrightarrow \sqrt{n}x + 1 + y \in S$. $B(x,y)$ is defined similarly using T instead.

We want to solve whether $\sum_{(x,y)} A(x,y) \cdot B(x,y) = 0$.[1] Suppose q is a **prime number** around n^3. Instead of working with discrete indicator function $A(x,y)$, we lift it to a polynomial over a **finite field** $\mathbb{F}_q$. Defining $\tilde{A}(x,y) : \mathbb{F}_q \times \mathbb{F}_q \to \mathbb{F}_q$, such that $\forall x, y \in [\sqrt{n}] \times [\sqrt{n}], A(x,y) = \tilde{A}(x,y)$. Then we design a polynomial over the field, that

One example of a polynomial that satisfies this is:

$$\tilde{A}(X,Y) = \sum_{x,y=1}^{\sqrt{n}} A(x,y) \cdot \frac{\prod_{x',y' \neq x,y} (X - x')(Y - y')}{\prod_{x',y' \neq x,y} (x - x')(y - y')}, \quad \text{with degree} \leqslant \sqrt{n}.$$

Now, instead of checking $\sum_{(x,y)} A(x,y) \cdot B(x,y) = 0$, we check $\sum_{x,y=1}^{\sqrt{n}} \tilde{A}(x,y)\tilde{B}(x,y)$ using Polynomial Identity Testing. Define $S(X)$ for Bob as:

$$S(X) = \sum_{y=1}^{\sqrt{n}} \tilde{A}(X,y) \cdot \tilde{B}(X,y).$$

The procedures of the communication will be the following:

1. Merlin will send $\tilde{S}$, which is a polynomial with degree $\sqrt{n}$ ($\sqrt{n}\log q$)bits), to Bob.
2. Alice will randomly choose $r \in \mathbb{F}_q$ and send r as well as $(\tilde{A}(r,1), \tilde{A}(r,2), \ldots, \tilde{A}(r,\sqrt{n}))$ (using $\sqrt{n}\log q$ bits), to Bob.
3. Bob can now compute $S(r)$ using information received from Alice and checks if $\tilde{S}(r) = S(r)$. If $\tilde{S}(r) = S(r)$, then by Schwartz-Zippel Lemma (Lemma 1), with high probability $\tilde{S} = S$ also. Thus, Bob can use $\tilde{S}(X)$ to make the correct conclusion with high probability, namely, check that if at least one of $S(1), \ldots, S(\sqrt{n})$ is non-zero.

By following this, we will get the desired result of our protocol. □

References

[AW08] Scott Aaronson and Avi Wigderson. Algebrization: a new barrier in complexity theory. In Cynthia Dwork, editor, *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 731–740. ACM, 2008. 4

[MVV87] Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. In Alfred V. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 345–354. ACM, 1987. 2

[Sch80] J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, October 1980. 1

[Zip79] Richard Zippel. Probabilistic algorithms for sparse polynomials. In *International symposium on symbolic and algebraic manipulation*, pages 216–226. Springer, 1979. 1

[1] In other words, if we have elements in both S and T, we get non-zero results, otherwise, we will receive 0.