**CMPUT 498/501: Advanced Algorithms** **Fall 2025**

# Lecture 11 (Oct 8, 2025): Algebraic Techniques

*Lecturer: Mohammad R. Salavatipour* *Scribe: Priyanshu Thakkar, Bhavya Thakkar*

## 11.1 Finger printing

Imagine a company maintaining multiple copies of a huge data set. From time to time, they would like to check whether these copies are identical or not. We can think of a data set as a binary string of length $n$. So the basic idea is simple : to compare two items $x$,$y$ from a huge universe $U$.

As an example consider the following setting where Alice Alice has a binary string $a_1, a_2, \ldots, a_n$ and Bob has a binary string $b_1, b_2, \ldots, b_n$. They would like to check whether their strings are equal with as few communications as possible.

**Deterministic Algorithm:** An obvious algorithm is for Alice to send all her bits to Bob, who can then check and tell Alice whether their strings are equal.

However, this approach requires $n$ bits of communication ($|x| = |y| = n$). As you might imagine, there are no better ways to do it, and it can be proved that there are no deterministic algorithms that can do better, using techniques developed in communication complexity.

**Randomized Algorithm:** There is a randomized algorithm for this problem that uses only $O(\log n)$ bits of communication.

We will be working over some unspecified field $\mathbb{F}$. Part of the reason we do not explicitly specify the underlying field is that typically the randomization will involve uniform sampling from a finite subset of the field; in such cases, we do not have to worry about whether the field is finite or not. It may be helpful to think of $\mathbb{F}$ as the field $\mathbb{Q}$ of the rational numbers; when we restrict ourselves to finite fields, it may be useful to assume that $\mathbb{F}$ is $\mathbb{Z}_p = \{0, 1, 2, \ldots, p-1\}$, the field of integers modulo some prime number $p$.

Suppose $x = a_1, a_2, \ldots, a_n$ and $y = b_1, b_2, \ldots, b_n$. We consider the polynomials

$$A(x) = \sum_{i=1}^{n} a_i x^i \quad \text{and} \quad B(x) = \sum_{i=1}^{n} b_i x^i.$$

**Algorithm 1** Consistency Check between Alice and Bob

Alice and Bob agree on a sufficiently large field $\mathbb{F}$ (e.g., modular arithmetic over a large prime $p$)
Alice picks a random element $r \in \mathbb{F}$, evaluates $A(r)$, and sends $r$ and $A(r)$ to Bob
Bob computes $B(r)$
**if** $A(r) = B(r)$ **then** return "consistent"
**else** return "inconsistent"

**Analysis :** If the two strings are equal, then $A(x) = B(x)$ and the algorithm will always output "consistent". Therefore, whenever it returns "inconsistent", it is guaranteed to be correct.

The only possibility for error occurs when the strings are different but the algorithm still outputs "consistent". This happens if the randomly chosen $r$ happens to be a root of $(A-B)(x)=0$.

Since $A$ and $B$ are polynomials of degree at most $n$, $(A-B)$ also has degree at most $n$ and thus has at most $n$ roots. Picking $r$ uniformly at random from $\mathbb{F}$, the probability that $r$ is a root is at most $n/p=\varepsilon$.

By choosing a large enough prime, say $p=n^2$, the algorithm achieves communication complexity of $2\log n$ bits and an error probability of at most $1/n$.

## 11.2 Finger printing : Frievald's technique

Consider the following problem: given matrices $A$, $B$, and $C$, we want an efficient way to verify if $AB=C$. The straightforward brute-force check requires $O(n^3)$ time, dominated by the multiplication of $A$ and $B$ (though advanced algorithms can reduce this to $O(n^{2.376\ldots})$).

We aim for a faster and simpler method to check $AB=C$, which is especially useful as a verification step after performing complex matrix multiplications.

Consider a random vector $r\in\{0,1\}^n$. Instead of computing the full product $AB$, we can check if $ABr=Cr$ as follows:

- Compute $x=Br$ in $O(n^2)$ time.
- Compute $y=Ax$ in $O(n^2)$ time.
- Compute $z=Cr$ in $O(n^2)$ time.

If AB = C there is no problem with this procedure, but what if AB is not equal to C but ABr = Cr? To rephrase, let $D=AB-C$. What if $Dr=0$ but $D\neq 0$?

If $D$ is non-zero, there must exist a non-zero row vector $\vec{d}$. Without loss of generality, assume $\vec{d}$ is the first row of $D$ that is not zero.

Then, for $Dr=0$ we have:

$$\sum_{i=1}^{n} d_i r_i = 0 \quad \Longrightarrow \quad r_1 = \frac{-\sum_{i=2}^{n} d_i r_i}{d_1}.$$

Assume that all $r_i$ for $i\geq 2$ are chosen before $r_1$. Then there is a unique value $v\in F$ for $r_1$, and the probability that $r_1$ takes that value is at most $1/2$.

We can repeat this process $k$ times to reduce the error probability to at most $1/2^k$.

## 11.3 Polynomial Identity Testing

Freivalds' technique is fairly general and can be applied to the verification of several different kinds of identities. In this section, we see that it also applies to the verification of identities involving polynomials.

Given polynomials $P$, $Q$, and $R$ over $\mathbb{F}$, we want to check if $P(x)Q(x)=R(x)$, where $P$ and $Q$ have degree at most $n$, and $R$ has degree at most $2n$. Using FFTs, we can multiply and evaluate $R(x)$ in $O(n\log n)$ time.

**A faster and randomized test :**

- Let $S \subseteq \mathbb{F}$, and pick $r \in S$ uniformly at random.
- Compute $P(r)Q(r) - R(r)$.
- If the result is non-zero, we know that the original equality does not hold.
- Otherwise, we conclude that the equality holds.

**What is the probability of error?**

If $D(x) = P(x)Q(x) - R(x)$ is not identically zero but the value $r$ that we chose is one of its roots, an error can occur.

Since the degree of $R(x)$ is at most $2n$, the degree of $D(x)$ is $\leq 2n$, and thus it has at most $2n$ roots. Therefore, the probability of error is $\leq 2n/|S|$.

**General Setting :**

Now, we consider a generalization of the argument to multivariate polynomials composed of $n$ variables. The degree of a term is defined as the sum of the degrees of the included variables. For example, $\deg(x^2yz^3) = 6$.

We are given a multivariate polynomial $P(x_1, \ldots, x_n)$ and the objective is to determine if $P(x_1, \ldots, x_n) = 0$, i.e., whether it is equal to the zero polynomial.

If the polynomial is given explicitly (e.g., $P(x_1, x_2, x_3) = x_1x_2^2x_3^3 + 4x_1^6x_3 + x_2x_3^2$), then this problem is straightforward. However, the polynomial can also be given compactly. For e.g.,

$$P(x_1, x_2, x_3) = (x_1 - x_3^2)(x_3^3 + x_2^4) \cdots (x_1 + x_2x_3)$$

or

$$P(x_1, x_2, x_3) = \det \begin{pmatrix} x_1 & x_3 & x_2 + x_4^2 \\ 0 & x_2 - x_3 & 1 \\ x_2^3 & x_1 & x_4x_2x_3 \end{pmatrix}.$$

, and the problem becomes difficult.

In some cases, we may only have access to a polynomial through a black box that can evaluate it at a given point.

Currently, there is no known deterministic polynomial-time algorithm for checking the identity of multivariate polynomials.

**Theorem (Schwartz and Zippel) :** Let $Q(x_1, \ldots, x_n)$ be a multivariate polynomial of total degree $d$. Let $S \subset F$, and let $r_1, r_2, \ldots, r_n$ be selected uniformly at random from $S$. Then

$$\mathbb{P}\left[Q(r_1, \ldots, r_n) = 0 \mid Q(x_1, \ldots, x_n) \not\equiv 0\right] \leq \frac{d}{|S|}.$$

**Proof :** We prove by induction on $n$. Base Case: $n = 1$, was done in the previous subsection.

Consider $Q$ and pick one variable, say $x_1$, that affects $Q$ and factor it:

$$Q = \sum_{i \leq k} x_1^i Q_i(x_2, \ldots, x_n)$$

where $k$ is the largest exponent of $x_1$. We have $Q_k(x_2, \ldots, x_n) \neq 0$ by the choice of $x_1$ and the total degree of $Q_k$ is at most $d - k$. First assume that $Q_k(r_2, \ldots, r_n) \neq 0$. Let $q(x_1) = Q(x_1, r_2, \ldots, r_n) = \sum_{i=0}^{k} x_1^i Q_i(r_2, \ldots, r_n)$. Since $q(x_1)$ is a single-variable polynomial, the probability that $q(x_1)$ equals zero is at most $\frac{k}{|S|}$. Also, by induction hypothesis and since total degree of $Q_k$ is at most $d - k$:

$$\mathbb{P}[Q_k(r_2, \ldots, r_n) = 0] \leq \frac{d-k}{|S|}.$$

Thus:

$$\mathbb{P}[Q_k(r_2, \ldots, r_n) = 0] \leq \frac{d-k}{|S|} \quad \text{and} \quad \mathbb{P}[Q(r_1, \ldots, r_n) = 0 \mid Q_k(r_1, \ldots, r_n) \neq 0] \leq \frac{k}{|S|}.$$

For any two events $E_1$ and $E_2$:

$$\mathbb{P}[E_1] = \mathbb{P}[E_1 \cap E_2] + \mathbb{P}[E_1 \cap \overline{E_2}] \leq \mathbb{P}[E_1 \mid \overline{E_2}] + \mathbb{P}[E_2].$$

Thus:

$$\mathbb{P}[Q(r_1, \ldots, r_n) = 0] \leq \frac{d-k}{|S|} + \frac{k}{|S|} = \frac{d}{|S|},$$

as claimed.

## 11.4 Perfect Matchings

Here, we discuss fast randomized methods for determining whether a bipartite graph contains a perfect matching and for constructing one if it exists. These methods make use of polynomial identity testing, as introduced earlier.

A bipartite graph $G(U \cup V, E)$ is a graph whose vertices are divided into two separate sets $U$ and $V$, with every edge connecting a vertex from $U$ to a vertex from $V$. When $|U| = |V| = n$, a *perfect matching* $M \subseteq E$ is a collection of $n$ edges such that each vertex is incident to exactly one edge in $M$, i.e., the subgraph formed by $M$ is 1-regular.

**Definition 1** *A **Tutte matrix** of a bipartite graph $G(U \cup V, E)$ is an $n \times n$ matrix $M$ defined as*

$$M_{ij} = \begin{cases} X_{ij} & \text{if } (u_i, v_j) \in E, \\ 0 & \text{otherwise.} \end{cases}$$

The determinant of this Tutte matrix provides a way to check if the graph contains a perfect matching.

**Theorem 1 (Edmond)** *If $\det(M) \neq 0$, then the bipartite graph $G$ has a perfect matching.*

**Proof.** By the definition of the determinant,

$$\det(M) = \sum_{\pi \in Pn} (-1)^{\text{sign}(\pi)} \prod_{i=1}^{n} M_{i,\pi(i)},$$

where $Pn$ is the set of all permutations of $\{1, \ldots, n\}$. There is a natural one-to-one correspondence between perfect matchings in $G$ and permutations of $\{1, \ldots, n\}$ given by

$$\{(v_1, u_{\pi(1)}), \ldots, (v_n, u_{\pi(n)})\}.$$

Hence, every term in the determinant expansion is nonzero if and only if it corresponds to a perfect matching of $G$. Since each product in the sum is unique, no two terms cancel out. ■

**Randomized Matching Algorithm:** This provides a simple method to check for the existence of a perfect matching in $G$.

Using Edmonds' theorem and the Schwartz-Zippel lemma, we can efficiently test for the existence of a perfect matching by evaluating whether $\det(M) = 0$:

- If $G$ has no perfect matching, then $\mathbb{P}[\text{accept}] = 0$.
- If $G$ has a perfect matching, then $\mathbb{P}[\text{accept}] \geq \frac{1}{2}$.

**Time complexity:** The most computationally intensive step of this algorithm is computing the determinant of the Tutte matrix. When the matrix entries are actual values from a field, the determinant can be computed using Gaussian elimination in $O(n^3)$ time. There also exist fast matrix multiplication algorithms with exponent $\omega \approx 2.371$, which can theoretically improve the runtime.

This approach was the fastest known algorithm for finding perfect matchings in dense bipartite graphs (as of 2022, faster algorithms have been developed).

**How to find a perfect matching?**

Given an algorithm to check for the existence of a perfect matching, we can construct a method to actually find one:

**Algorithm 2** Finding a Perfect Matching

```
Pick an edge u_i v_j ∈ E and denote it as e
if G − e has a perfect matching then
    Include edge e in the matching
    Recurse on G − e
else
    Discard edge e
    Recurse on G − e
end if
```

## 11.5 Parallel Algorithm and Isolation Lemma

Another benefit of the algebraic approach to perfect matchings is that many of its steps can be performed efficiently in parallel. In particular,

**Theorem 2** *The determinant of an $n \times n$ matrix can be computed in $O(\log^2 n)$ time using $O(n^{\omega+2})$ processors.*

By combining this result with the algebraic matching algorithm, we can design an efficient parallel method to determine whether a graph has a perfect matching (YES or NO).

### 11.5.1 Checking for a Perfect Matching in Parallel

To check for a perfect matching in parallel, we can assign each processor to remove one edge $e = uv$ and verify whether $G - e$ still contains a perfect matching. However, the main difficulty lies in coordinating all processors to focus on the same matching, since a graph may have exponentially many perfect matchings. The *Isolation Lemma* provides an elegant way to handle this issue.

**Lemma 1 (Isolation Lemma)** *Let $X = \{1, \ldots, m\}$ be a set of elements, and let $\mathcal{F} = \{S_1, \ldots, S_k\}$ be a family of distinct subsets of $X$. Assign each element $x_i \in X$ an independent random weight $w(x_i)$ chosen uniformly from $\{1, \ldots, 2m\}$. Then,*

$$\mathbb{P}[\textit{the minimum-weight set in } \mathcal{F} \textit{ is unique}] >= \frac{1}{2}.$$

**Remark.** At first glance, this result may seem counterintuitive because the number of subsets $k$ can be as large as $2^m$. Since each subset has a total weight in the range $\{1, \ldots, 2m^2\}$, one might expect several subsets to share the same minimum weight. However, the lemma guarantees that with probability greater than $1/2$, there will be exactly one subset with the smallest total weight.

**Proof.** Let us fix an element $x_i$ and define the following:

$$\mathcal{Y}_i^+ : \text{the family of all sets in } \mathcal{F} \text{ that include } x_i$$

$$\mathcal{Y}_i^- : \text{the family of all sets in } \mathcal{F} \text{ that exclude } x_i$$

$$\alpha_i = \min_{S \in \mathcal{Y}_i^-} w(S) - \min_{S' \in \mathcal{Y}_i^+} w(S' - \{x_i\})$$

Observe that if $\alpha_i < w(x_i)$, then $x_i$ cannot appear in any minimum-weight set, since there exists a set $S' \in \mathcal{Y}_i^+$ with smaller weight than $w(S' - \{x_i\})$. Hence, for any $S \in \mathcal{Y}_i^-$,

$$w(S) < w(S' - \{x_i\}) + w(x_i) = w(S').$$

Conversely, when $\alpha_i > w(x_i)$, every minimum-weight set must contain $x_i$.

We call $x_i$ **ambiguous** if $\alpha_i = w(x_i)$.

Note that $\alpha_i$ does not depend on $w(x_i)$, and $w(x_i)$ is sampled uniformly from $\{1, \ldots, 2m\}$. Therefore,

$$\mathbb{P}[x_i \text{ is ambiguous}] \leq \frac{1}{2m}.$$

Using the union bound, we get

$$\mathbb{P}[\text{some } x_i \text{ is ambiguous}] \leq \frac{1}{2}.$$

Thus, when no element is ambiguous, the minimum-weight set is guaranteed to be unique. ■

**Matching using the Isolation Lemma:**

Let $x \in E$ and let $\mathcal{F}$ be the set of all perfect matchings. By the Isolation Lemma, if we assign random weights from $\{1, \ldots, 2m\}$ to the edges, then with probability greater than $\frac{1}{2}$, the minimum-weight perfect matching is unique.

Let $X_{ij} = 2^{w_{ij}}$, where $w_{ij}$ is the weight of the edge $u_i v_j$. Define $B$ as the modified Tutte matrix whose entries are given by these weighted terms.

**Lemma 18.8:** If there exists a unique minimum-weight perfect matching in $G$ with total weight $W$, then:

(i) $\det(B) \neq 0$, and

(ii) the largest power of 2 dividing $\det(B)$ is $2^W$.

**Proof:** All other perfect matchings (apart from the unique minimum-weight one) have total weights in $\{W + 1, W + 2, \ldots\}$. Thus, each term in $\det(B)$ has the form $\pm 2^W, \pm 2^{W+1}, \ldots$

Factor out $2^W$ from all terms:

$$\det(B) = 2^W (1 + a_1 \cdot 2^1 + a_2 \cdot 2^2 + \ldots)$$

where each $a_i$ is an integer (possibly negative or zero). Since the sum inside the parentheses is odd, it is nonzero, implying that $\det(B) \neq 0$. Hence, $2^W$ is the largest power of 2 dividing $\det(B)$.

**Algorithm 3** Parallel Algorithm for Matching

```
Choose each w_ij uniformly at random from {1, ..., 2m}, where m = |E|
Construct the Tutte matrix B using the w_ij values
Compute det(B) in parallel, and determine the largest W such that 2^W divides det(B)
if det(B) = 0 then
  return no perfect matching
end if
for each u_i v_j ∈ E in parallel do
  Compute det(B_ij), where B_ij is the minor obtained by deleting row i and column j
  if det(B_ij) ≠ 0 then
    Find the largest power of 2, say 2^W', that divides det(B_ij)
    if W' + w_ij = W then
      Output edge u_i v_j as part of the perfect matching
    end if
  end if
end for
```

# References


[1] Rajeev Motwani and Prabhakar Raghavan, *Randomized Algorithms*, Cambridge University Press. Available at: `https://rajsain.files.wordpress.com/2013/11/randomized-algorithms-motwani-and-raghavan.pdf`

[2] Sepehr Assadi, *CS 466/666: Randomized Algorithms (Fall 2023), Rutgers University.* Available at: `https://sepehr.assadi.info/courses/cs466(6)-f23/#FT87`

[3] Mohammad R. Salavatipour, *CMPUT 675: Randomized Algorithms (Fall 2005)*, University of Alberta. Available at: `https://webdocs.cs.ualberta.ca/%7Emreza/courses/Random05/index.html`