COMS 6998: Algebraic Techniques in TCS (Fall'21) Sep 14, 2021

Lecture 1: Introduction, Graph algorithms using MM

Instructor: *Josh Alman* Scribe notes by: *Shunhua Jiang*

Disclaimer: This draft may be incomplete or have errors. Consult the course webpage for the most up-to-date version.

1 Logistics

Prerequisites.

- Mathematical maturity: read and write formal math proofs.
- Design and analysis of algorithms.
- Linear algebra.

Grading.

- Scribe notes (15%): Scribe for one lecture. Draft due two days after class.
- Problem sets (35%): 3 or 4 in total. Each due in two weeks.
- Final project (50%): Choose between (1) A reading-based project to survey one or more papers. (2) A research project.

  Final report of 5-15 pages + presentation.

2 Overview

We will study applications of algebraic techniques in TCS. Here "TCS" mainly means algorithms and complexity. We will briefly mention other areas as well, e.g., using the polynomial method in learning theory. This course mainly covers four topics:

- Algebraic graph algorithms.
- The polynomial method.
- Matrix rigidity.
- Matrix multiplication.

2.1 Algebraic Graph Algorithms

We study algebraic tools for faster graph algorithms. Some examples:

- **Graph problems:** All-pairs shortest paths (APSP), subgraph isomorphism, maximum matching, longest path.
- **Algebraic tools:** Polynomial identity testing, fast matrix multiplication (FMM), algorithms for determinant/inverse.

2.2 The Polynomial Method

The polynomial method represents the target Boolean function as a polynomial, and then derives properties of the Boolean function by studying the polynomial.

Two ways that we will measure the complexity of a polynomial are:

1. **Sparsity:** number of monomials in the polynomial,
2. **Degree:** maximum degree of all monomials.

2.2.1 Examples of Polynomials Representing Boolean Functions

Exact representation of the AND function. The Boolean AND function $AND : \{0,1\}^n \to \{0,1\}$ is defined as

$$AND(x_1, x_2, \cdots, x_n) = \begin{cases} 1, & \text{if } x_1 = x_2 = \cdots = x_n = 1, \\ 0, & \text{otherwise.} \end{cases}$$

Define a polynomial $p : \mathbb{R}^n \to \mathbb{R}$ as $p(x_1, x_2, \cdots, x_n) = x_1 \cdot x_2 \cdots x_n$. It's easy to see that

$$\forall x \in \{0,1\}^n,\ p(x) = AND(x).$$

The polynomial p has sparsity$(p) = 1$ and $\deg(p) = n$.

Question: Does there exist a polynomial that equals to the AND function on all $x \in \{0,1\}^n$ and has degree $< n$? The answer is no. See the homework.

Instead, we can loosen the restrictions on what it means for a polynomial to compute a Boolean function to try to achieve lower degree.

Polynomial threshold function. We say a polynomial $q : \mathbb{R}^n \to \mathbb{R}$ is a polynomial threshold function for AND if it satisfies the following: $\forall x \in \{0,1\}^n$,

$$\begin{aligned} &\text{if } AND(x) = 1,\ \text{then } q(x) \geq 0, \\ &\text{if } AND(x) = 0,\ \text{then } q(x) < 0. \end{aligned}$$

For example, the following polynomial q is a polynomial threshold function for the AND function:

$$q(x) = x_1 + x_2 + \cdots + x_n - \left(n - \frac{1}{2}\right).$$

This is because for $x_1 = x_2 = \cdots = x_n = 1$, $q(x) = n - \left(n - \frac{1}{2}\right) = \frac{1}{2} \geq 0$. And if some $x_i = 0$, $q(x) \leq (n-1) - (n - \frac{1}{2}) = -\frac{1}{2} < 0$.

Note that the degree of q is $\deg(q) = 1$. This is much smaller than the degree of the previous polynomial $p(x) = x_1 \cdot x_2 \cdots x_n$.

Approximate polynomial. We say a polynomial $q : \mathbb{R}^n \to \mathbb{R}$ is an approximate polynomial for a Boolean function $f : \{0,1\}^n \to \{0,1\}$ if it satisfies the following: $\forall x \in \{0,1\}^n$,

$$\text{if } ANDD(x) = 1, \text{ then } q(x) = 1,$$
$$\text{if } ANDD(x) = 0, \text{ then } -\frac{1}{3} \leq q(x) \leq \frac{1}{3}.$$

We can design an approximate polynomial for the AND function with degree $\Theta(\sqrt{n})$ using Chebyshev polynomials. See homework.

Other types of approximate polynomials. Another definition of approximate polynomial: $q(x) = f(x)$ for at least $\frac{3}{4}$ fraction of $x \in \{0,1\}^n$. The polynomial $q(x) = 0$ is such an approximation for the AND function since $ANDD(x) = 0$ for all x except the all one vector.

What if we further add another condition that $q(x)$ must equal to 1 when $f(x) = 1$? The polynomial $q(x) = x_1 \cdot x_2$ satisfies this stronger approximation condition for the AND function. This is because $q(\mathbf{1}_n) = 1$, and only $\frac{1}{4}$ fraction of $x \in \{0,1\}^n$ has $x_1 = x_2 = 1$.

2.2.2 Applications

1. **Lower bounds.** The polynomial method has been used to prove lower bounds in circuit complexity and communication complexity, e.g., circuit lower bounds for AC^0.

   The high-level idea is to use a reduction from the known lower bounds of the degree of polynomials.

2. **Faster algorithms.** The polynomial method is also useful for designing faster algorithms, e.g., nearest neighbor search (NNS), all-pairs shortest paths (APSP).

   The high-level idea behind these algorithms is to first convert the original problem into a Boolean function, then approximate the Boolean function using a polynomial with good properties, and finally design algorithms to evaluate the polynomial efficiently.

2.3 Matrix Rigidity

Definition 1 (Matrix rigidity)**.** *Given an integer r and a matrix M of size $N \times N$, the rank r rigidity of M, denoted as $\mathscr{R}_M(r)$, is the minimum number of entries of M one must change to make its rank $\leq r$.*

2.3.1 Examples

Identity matrix. The first example is the simplest identity matrix:

$$I_N = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{bmatrix} \Rightarrow \left[\begin{array}{c|ccc} 1 & & & \\ \hline & 0 & & \\ & & 0 & \\ & & & 0 \end{array}\right].$$

N $\qquad$ r $\quad$ $N-r$

We can change $N-r$ diagonal entries of I_N to zero to decrease its rank to r. Thus $\mathscr{R}_{I_N}(r) \leq N-r$.

In fact, $\mathscr{R}_{I_N}(r) = N-r$. This is because changing one entry of a matrix can decrease its rank by at most 1, so we need to change at least $N-r$ entries of I_N until its rank decreases to r.

Upper triangular matrix. A more rigid example is the upper triangular matrix:

$$U_N = \begin{bmatrix} 1 & 1 & 1 & 1 \\ & 1 & 1 & 1 \\ & & 1 & 1 \\ & & & 1 \end{bmatrix}.$$

A naive upper bound is $\mathscr{R}_{U_N}(r) \leq \sum_{i=1}^{N-r} i = O((N-r)^2)$ where we change all the ones in the lower $N-r$ rows to zero.

Here is a more efficient way to decrease the rank. We divide the rows of U_N into r groups of size $\frac{N}{r}$, and in each group we change the $\sum_{i=1}^{N/r-1} i = O((\frac{N}{r})^2)$ number of zeroes in the bottom left corner to one. In this way all the rows in one group become the same, so the rank of the modified matrix is r. Below we show an illustration for $N=6$ and $r=2$:

$$\left[\begin{array}{cccccc} 1 & 1 & 1 & 1 & 1 & 1 \\ * & 1 & 1 & 1 & 1 & 1 \\ * & * & 1 & 1 & 1 & 1 \\ \hline & & & 1 & 1 & 1 \\ & & & * & 1 & 1 \\ & & & * & * & 1 \end{array}\right] \quad r \text{ groups of size } \frac{N}{r}.$$

We get a better upper bound $\mathscr{R}_{U_N}(r) \leq O(r \cdot (\frac{N}{r})^2) = O(\frac{N^2}{r})$. This bound is actually tight.

2.3.2 Valiant Rigidity

Is the upper triangular matrix rigid enough? What is the standard of rigidity? A gold standard is the Valiant rigidity [Val77]: A matrix M of size $N \times N$ is Valiant-rigid if

$$\mathscr{R}_M\left(\frac{N}{\log\log N}\right) \geq N^{1+\epsilon} \text{ for some constant } \epsilon > 0.$$

The upper triangular matrix is far from being Valiant-rigid:

$$\mathscr{R}_{U_N}\left(\frac{N}{\log\log N}\right) = O\left(\frac{N^2}{N/\log\log N}\right) = O(N\log\log N) \ll N^{1+\epsilon}.$$

In [Val77], Valiant showed that if there exists an *explicit* construction of a family of matrices that is Valiant-rigid, then we can prove a big break-through result in circuit complexity. Currently we still do not have any explicit construction of matrices that are Valiant-rigid.

Meanwhile, it is known that a uniformly random $\{0,1\}$-matrix M satisfies

$$\mathscr{R}_M\left(\frac{N}{2}\right) \geq \Omega(N^2)$$

with high probability, and this is even stronger than Valiant-rigidity.

Thus, even though most of the $\{0,1\}$-matrices satisfy our desired property, we still do not have any explicit construction. This is a common phenomenon in TCS, often referred to as "finding hay in a haystack".

2.4 Matrix Multiplication

We are given two matrices $A, B \in \mathbb{F}^{N\times N}$ over some field $\mathbb{F}$ as input, and the goal is to compute $C = A\times B$, i.e., $C_{ij} = \sum_{k=1}^{N} A_{ik}\cdot B_{kj}$.

Note that naively computing the matrix multiplication takes $O(N^3)$ operations.

We always measure the complexity of matrix multiplication by the number of *arithmetic operations*. Rather than saying $O(\cdots)$ time, we will say $O(\cdots)$ operations. This is because for different field $\mathbb{F}$, the time to multiply/add two entries of A and B can be very different. The time to multiply two matrices is then the number of operations multiplied by the time to do each operation over the field.

2.4.1 Strassen's Algorithm

In 1969, Strassen [Str69] published the first matrix multiplication algorithm that runs in $o(N^3)$ time. His algorithm runs in $O(N^{2.81})$ time.

Computing 2×2 matrix multiplication faster. Let A and B be two matrices of size 2×2:

$$A = \left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right],\ B = \left[\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array}\right].$$

When computing $A\times B$, the naive algorithm uses **4 additions** and **8 multiplications**.

Strassen showed a way to compute $A\times B$ using **18 additions** and **7 multiplications**. Strassen's algorithm saves the number of multiplications by increasing the number of additions.

For completeness we include Strassen's algorithm for 2×2 matrices here. First compute M_1 to M_7:

$$\begin{aligned}
M_1 &= (A_{11} + A_{22})(B_{11} + B_{22}), \\
M_2 &= (A_{21} + A_{22})B_{11}, \\
M_3 &= A_{11}(B_{12} - B_{22}), \\
M_4 &= A_{22}(B_{21} - B_{11}), \\
M_5 &= (A_{11} + A_{12})B_{22}, \\
M_6 &= (A_{21} - A_{11})(B_{11} + B_{12}), \\
M_7 &= (A_{12} - A_{22})(B_{21} + B_{22}).
\end{aligned}$$

The matrix $C = A \times B$ is then computed as:

$$\begin{aligned}
C_{11} &= M_1 + M_4 - M_5 + M_7, \\
C_{12} &= M_3 + M_5, \\
C_{21} &= M_2 + M_4, \\
C_{22} &= M_1 - M_2 + M_3 + M_6.
\end{aligned}$$

The recursive algorithm. For larger matrices, Strassen's algorithm divides the matrix into 4 block matrices of size $\frac{N}{2} \times \frac{N}{2}$, and computes the multiplications of the submatrices recursively. In order to multiply two $N \times N$ matrices, the algorithm plugs these submatrices into Strassen's identity above, so it computes 18 additions of $\frac{N}{2} \times \frac{N}{2}$ submatrices, and 7 multiplications of $\frac{N}{2} \times \frac{N}{2}$ submatrices. Fortunately, even though we are doing many additions, we can very quickly add matrices in just $O(N^2)$ time. Thus we get the following recursive formula for the running time:

$$\begin{aligned}
T(N) &= 7 \cdot T\left(\frac{N}{2}\right) + 18 \cdot O(N^2) \\
\implies T(N) &= O(N^{\log_2 7}) \leq O(N^{2.81}).
\end{aligned}$$

Current fast matrix multiplication algorithms. Currently, the fastest matrix multiplication algorithm runs in $O(N^{2.373})$ time.

Even though Strassen's algorithm is used in practice, the later theoretically faster algorithms usually have exceedingly large constants, and they cannot be used in practice.

2.4.2 Applications

1. Matrix multiplication is used in all three previous problems.

   In this course, we will first use matrix multiplication as a black-box to design algorithms for other problems, and in the end we cover the fast matrix multiplication algorithms.

2. Many other linear algebra tasks can be done in the same time as matrix multiplication, including computing determinant, inverse, linear systems, and some linear programs.

3 Graph Algorithms Using MM

Now we delve into our first topic (algebraic graph algorithms). In this section we consider designing graph algorithms using the algebraic tool of matrix multiplication.

3.1 Finding triangles in a graph

Input: An undirected graph G on N nodes.
Output: Are there nodes a, b, c such that $(a, b), (b, c), (c, a) \in E(G)$?

We can trivially solve this problem in $O(N^3)$ time by enumerating all triples (a, b, c). Next we show an algorithm that runs in $O(N^{2.373})$ time (the matrix multiplication time).

Algorithm. The algorithm first forms the adjacency matrix $A \in \{0, 1\}^{N \times N}$ of G:

$$A_{ij} = \begin{cases} 1, & \text{if } (i, j) \in E(G), \\ 0, & \text{otherwise.} \end{cases}$$

The algorithm then computes the matrix A^2, and checks if there exists a pair (a, b) that satisfies

$$(a, b) \in E(G) \text{ and } A^2[a, b] > 0.$$

If so, the algorithm outputs "yes" (there exists a triangle), and otherwise the algorithm outputs "no".

Analysis. We first show the geometric interpretation of the matrix A^2:

$$\begin{aligned} A^2[i, j] &= \sum_{k=1}^{N} A[i, k] \cdot A[k, j] \\ &= \#\text{ of } k \text{ s.t. } (i, j) \text{ and } (k, j) \in E(G) \\ &= \#\text{ of length-2 paths from } i \text{ to } j. \end{aligned}$$

$A^2[a, b] > 0$ means there exists a length-2 path from a to b, i.e., there exists another node c where $(a, c), (b, c) \in E(G)$. Thus (a, b, c) is a triangle iff $(a, b) \in E(G)$ and $A^2[a, b] > 0$.

The bottleneck of this algorithm is to compute A^2, and this takes $O(N^{2.373})$ time. All other computation can be performed in $O(N^2)$ time.

3.2 Finding $\diamond$ in a graph

In this section we study another induced subgraph isomorphism problem. The target subgraph is shown in Figure 1.
Input: An undirected graph G on N nodes.
Output: Are there nodes a, b, c, d such that $(a, b), (b, c), (c, a), (a, d), (b, d) \in E(G)$ and $(c, d) \notin E(G)$?
An incorrect algorithm. Inspired by the triangle algorithm, a straightforward algorithm computes A^2 and checks if there exists (a, b) that satisfies

$$(a, b) \in E(G) \text{ and } A^2[a, b] \geq 2.$$

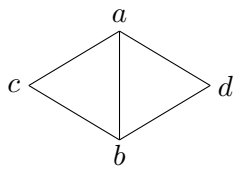

Figure 1: Target subgraph.

However, this naive algorithm doesn't work because it doesn't rule out the existence of the edge (c, d). A correct algorithm must distinguish our target subgraph $\diamond$ and the 4-clique $\boxplus$.

The correct algorithm. We make the following observation:

$$\sum_{(a,b)\in E(G)} \binom{A^2[a,b]}{2} = \#(\diamond) + \#(\boxplus) \cdot 6. \tag{1}$$

This is because $\sum_{(a,b)\in E(G)} \binom{A^2[a,b]}{2}$ counts the total number of pairs of "parallel" length-2 paths. A target subgraph $\diamond$ contributes 1 pair (the pair $(a - c - b)$ and $(a - d - b)$), while a 4-clique $\boxplus$ contributes $\binom{4}{2} = 6$ pairs (one pair between any two nodes in $\{a, b, c, d\}$).

Define $R(G) := \sum_{(a,b)\in E(G)} \binom{A^2[a,b]}{2}$. Eq. (1) implies the following:

- If $R(G)$ is not a multiple of 6, then the graph G must contain $\diamond$.
- If $R(G)$ is a multiple of 6, then it's not clear if G doesn't contain $\diamond$, or if the number of $\diamond$ in G is a multiple of 6.

In order to truly determine whether G contains $\diamond$, we design a randomized algorithm. The algorithm first samples a subgraph G' of G where G' keeps each node of G with probability $\frac{1}{2}$. In the next lecture, we will show that if G contains $\diamond$, then with high probability $R(G')$ is not a multiple of 6.

References

[Str69] Volker Strassen. Gaussian elimination is not optimal. *Numerische mathematik*, 13(4):354–356, 1969.

[Val77] Leslie G Valiant. Graph-theoretic arguments in low-level complexity. In *International Symposium on Mathematical Foundations of Computer Science*, pages 162–176. Springer, 1977.