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

Lecture 3: k-Paths, Matrix Inverse, and Fine-Grained Complexity

Instructor: *Josh Alman* Scribe notes by: *Yuhao Li, Thomas Tran*

1 Fixed-parameter tractability of $k-$Path

1.1 Problem Definition and Main Theorem

In this section, we focus on the following decision problem.

Problem 1 (k-Path)**.** *Given a directed graph G and a positive integer k, does there exist a path[1] of length k?*

Note that this k-Path problem is in NP, and is not hard to prove it is NP-hard by a reduction from the Hamiltonian path problem. Thus, rather than design a polynomial-time algorithm for the problem, we will aim to design a *fixed-parameter tractable algorithm* for the $k-$Path problem, which runs in polynomial time when k is a constant or $k \leq c \log n$ for a fixed constant c. Our main theorem is the following algorithmic result.

Theorem 1. *There is a randomized algorithm which, given as input a directed graph G on n nodes and a positive integer k, decides if there is a path of length k in G in $O(2^k \cdot \text{poly}(n,k))$ time, with arbitrarily small constant error probability.*

1.2 Reduction to multilinear monomial detection

Let W_k be the set of k-length walks in G. Consider the polynomial

$$p(x_1, \cdots, x_n) = \sum_{(i_0,\cdots,i_k)\in W_k} x_{i0}x_{i_1}\cdots x_{i_k}.$$

Then there is a k-length path in G if and only if there is a multilinear monomial in p. Before we see how to determine whether p has a multilinear monomial, we first note that there is an efficient algorithm for evaluating p.

Proposition 1. *For any field $\mathbb{F}$, given $x = (x_1, \cdots, x_n) \in \mathbb{F}^n$, we can compute $p(x)$ in $O(n^{2.373} \log k)$ field operations.*

Proof. Let $A \in \{0,1\}^{n\times n}$ be the adjacency matrix of G, and let $D \in \mathbb{F}^{n\times n}$ be the diagonal matrix such that $D[i,i] = x_i$ for all $i \in \{1,\ldots,n\}$. Then,

$$p(x_1, \cdots, x_n) = \textit{the sum of entries of } D(AD)^k,$$

[1]Recall the difference between a *walk* and a *path* in graph theory. A walk is a finite or infinite sequence of edges which joins a sequence of vertices. A path is a walk in which all edges and all vertices are distinct.

which could be computed in $O(n^{2.373} \log k)$ time by the exponentiation by squaring method (compute $(AD)^2$ first, then $(AD)^4$, $(AD)^8$, etc). □

1.3 Multilinear monomial detection

To help us to detect multilinear monomials in the polynomial p, we will consider a slightly different polynomial q. Pick a $(k+1) \times n$ matrix $A \in \mathbb{F}_2^{(k+1)\times n}$. Consider the new polynomial

$$q(x_1, \cdots, x_n) = \sum_{(i_0,\cdots,i_k)\in W_k} x_{i_0} x_{i_1} \cdots x_{i_k} \cdot \det(A_{(i_0,\cdots,i_k)}),$$

where $\det(A_{(i_0,\cdots,i_k)})$ is the determinant of the $(k+1) \times (k+1)$ matrix with the j-th column being the i_j-th column of A.

For any monomial $x_{i_0} x_{i_1} \cdots x_{i_k}$ in p that is not multilinear, there are at least two equal columns in the matrix $A_{i_0,\cdots,i_k}$, so the matrix does not have full rank and $\det(A_{(i_0,\cdots,i_k)}) = 0$.

For any monomial in p that is multilinear, we still cannot guarantee that $\det(A_{(i_0,\cdots,i_k)}) \neq 0$. However, the following lemma asserts that if we sample the matrix A uniformly at random from $\mathbb{F}_2^{(k+1)\times n}$, then for any monomial $x_{i_0} x_{i_1} \cdots x_{i_k}$ in p that is multilinear, the matrix $A_{(i_0,\cdots,i_k)}$ will have full rank with decent probability.

Lemma 2. *Pick a matrix $M \in \mathbb{F}_2^{(k+1)*(k+1)}$ uniformly at random. Then M has full rank with probability ≥ 0.28.*

Proof. For any $0 \leq i \leq k$, we use a_i to denote the i-th column of M.

Notice that M has full rank if, for all $i \in \{1, \cdots, k\}$, the vector a_i is not in $\text{span}\{a_0, \cdots, a_{i-1}\}$ and $a_0 \neq (0, \cdots, 0)$.

First,

$$\Pr[a_0 = (0, \cdots, 0)] = 1/2^{k+1}.$$

Now, suppose $a_0, \cdots, a_{i-1}$ are linearly independent. Then, since we are working over $\mathbb{F}_2$, we know that $|\text{span}\{a_0, \cdots, a_{i-1}\}| = 2^i$ and $|\mathbb{F}_2^{k+1}| = 2^{k+1}$, and so

$$\Pr[a_i \in \text{span}\{a_0, \cdots, a_{i-1}\}] = 2^i/2^{k+1}.$$

Hence,

$$\Pr[M \text{ has full rank}] = \prod_{i=0}^{k}(1 - 2^{i-k-1}) = \prod_{i=1}^{k+1}(1 - 2^i) > \prod_{i=1}^{+\infty}(1 - 2^i) > 0.28.$$ □

Algorithm. Using this lemma, the randomized algorithm in Theorem 1 works as follows: Sample a $(k+1) \times n$ matrix A from $\mathbb{F}_2^{(k+1)\times n}$ uniformly at random. If the polynomial p does not contain any multilinear monomial, then $q = 0$. If the polynomial p contains at least one multilinear monomial, then with probability at least 0.28 the polynomial q is a non-zero polynomial. We further repeat this process multiple times to achieve a larger success probability.

Now the only missing piece is to efficiently determine whether $q = 0$. Recall in the previous lecture (lecture 2) we proved the Schwartz-Zippel lemma, which is a probabilistic polynomial identity testing algorithm that can determine if $q = 0$ by evaluating q on some random inputs. So the only thing we need to deal with is evaluating $q(x_1, \cdots, x_n)$ efficiently on any given input $x = (x_1, \cdots, x_n)$.

1.4 Evaluating q efficiently

We first prove the following property of determinants.

Lemma 3. *If $M \in \mathbb{F}_2^{(k+1)\times(k+1)}$, then*

$$\det(M) = \sum_{b\in\mathbb{F}_2^{k+1}} \prod_{i=0}^{k} \langle M_i, b\rangle,$$

where M_i is the i-th column of M.

Proof. Notice that for any $b \in \mathbb{F}_2^{k+1}$, $\langle M_i, b\rangle$ is equal to 0 or 1. Then when $\prod_{i=0}^{k}\langle M_i, b\rangle = 1$, we must have $\langle M_i, b\rangle = 1$ for all $i \in \{0, \ldots, k\}$, i.e.,

$$Mb = \mathbf{1}_{k+1},$$

where $\mathbf{1}_{k+1} = (1, 1, \ldots, 1)^\top$.

If M has full rank, then there is exactly one b such that $Mb = (1, 1, \cdots, 1)^\top$. In this case, we know that $\det(M) = 1$, so $\det(M) = \sum_{b\in\mathbb{F}_2^{k+1}} \prod_{i=0}^{k}\langle M_i, b\rangle$ holds.

If M does not have full rank, then either there is no b such that $Mb = \mathbf{1}_{k+1}$, or (since we are working over $\mathbb{F}_2^{k+1}$) there are $2^{k+1-\mathrm{rank}(M)}$ different $b \in \mathbb{F}_2^{k+1}$ such that $Mb = \mathbf{1}_{k+1}$. So $\sum_{b\in\mathbb{F}_2^{k+1}} \prod_{i=0}^{k}\langle M_i, b\rangle$ must be 0 (over $\mathbb{F}_2$). Also because M does not have full rank, $\det(M) = 0$ follows. □

Using this property, we can evaluate q efficiently as follows.

$$\begin{aligned}
q(x_1, \cdots, x_n) &= \sum_{(i_0,\cdots,i_k)\in W_k} x_{i_0}x_{i_1}\cdots x_{i_k} \cdot \det(A_{(i_0,\cdots,i_k)}) \\
&= \sum_{(i_0,\cdots,i_k)\in W_k} x_{i_0}x_{i_1}\cdots x_{i_k} \cdot \sum_{b\in\mathbb{F}_2^{k+1}} \prod_{j=0}^{k} \langle a_{i_j}, b\rangle \\
&= \sum_{b\in\mathbb{F}_2^{k+1}} \sum_{(i_0,\cdots,i_k)\in W_k} \prod_{j=0}^{k} x_{i_j}\langle a_{i_j}, b\rangle \\
&= \sum_{b\in\mathbb{F}_2^{k+1}} p(x_1\langle a_1, b\rangle, \cdots, x_n\langle a_n, b\rangle).
\end{aligned}$$

In this way, we reduce the problem of evaluating q on one input to evaluating p on 2^{k+1} inputs. Using Proposition 1, this can be done in $O(2^k \cdot \mathrm{poly}(n, k))$ time. This finishes the proof of Theorem 1.

2 Fast Matrix Inverse

In this section, we aim to invert an $n \times n$ matrix over $\mathbb{R}$ in $O(n^{2.373})$ operations.

2.1 Simple Example

We first show how to invert a full-rank $n \times n$ upper triangular matrix M in any field. To do so, we first block M as

$$M = \left[\begin{array}{c|c} A & B \\ \hline 0 & C \end{array}\right]$$

where A, B, C are $n/2 \times n/2$ block matrices. Since M is a full-rank upper triangular matrix, we can see that A and C are also full-rank upper trianglular matrices.

We then make use of the following identity:

$$M^{-1} = \left[\begin{array}{c|c} A & B \\ \hline 0 & C \end{array}\right]^{-1} = \begin{bmatrix} A^{-1} & -A^{-1}BC^{-1} \\ 0 & C^{-1} \end{bmatrix}.$$

Algorithm. To invert an upper triangular matrix M,

1. Recursively invert A, C as they are also full-rank upper triangular matrices.
2. Compute $-A^{-1}BC^{-1}$.

Time Complexity. Let $T(n)$ denote the time to invert an $n \times n$ matrix. The two recursive calls take $2 \cdot T(\frac{n}{2})$ time, and computing $-A^{-1}BC^{-1}$ takes $O(n^{2.373})$ time. So we have the following recurrence:

$$T(n) = 2 \cdot T\big(\frac{n}{2}\big) + O(n^{2.373}).$$

By the Master Theorem, the time complexity of the algorithm is $O(n^{2.373})$.

2.2 General Case

To invert any $n \times n$ matrix M over $\mathbb{R}$, we make use of the following identity:

$$M = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right] = \begin{bmatrix} I & 0 \\ CA^{-1} & I \end{bmatrix} \begin{bmatrix} A & 0 \\ 0 & D - CA^{-1}B \end{bmatrix} \begin{bmatrix} I & A^{-1}B \\ 0 & I \end{bmatrix}.$$

Idea to compute M^{-1}: Invert each matrix individually, reverse the order, and multiply all three inverted matrices. Notice that the first matrix is lower triangular, the middle matrix's blocks can be inverted individually, and the last matrix is upper triangular. Using a similar recurrence formula as before, we can show that inverting M takes $O(n^{2.373})$ time. **The caveat here is that this assumes the top left block, A, is invertible**.

To remedy this, we will instead invert $M^\top M$, since its top-left block is always full rank (see Lemma 4 below for the proof), and use it to compute M^{-1}.

Algorithm. To invert M over $\mathbb{R}$ using the inverse of $M^\top M$,

1. Invert $M^\top M$.
2. Compute $(M^\top M)^{-1}M^\top = M^{-1}(M^\top)^{-1}M^\top = M^{-1}$.

We remark that for other fields there are similar but more complicated techniques to make A invertible.

Lemma 4. *If $M \in \mathbb{R}^{n \times n}$ is full rank, the top left block A of $M^\top M$ has full rank.*

Proof. Assume to the contrary that A of $M^\top M$ is not full rank. Thus there is a nonzero vector $x \in \mathbb{R}^{n/2}$ s.t. $Ax = 0$. Now, form $y \in \mathbb{R}^n$ to be the x vector in the first $n/2$ dimensions, and zero for the rest:

$$y = \begin{bmatrix} x \\ 0 \end{bmatrix}.$$

Thus, by definition of y, we have the following equality:

$$y^\top \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right] y = x^\top A x.$$

And by our assumption that $Ax = 0$,

$$y^\top \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right] y = 0.$$

However, we know that $y^\top M^\top M y = v^\top v$, where $v = My \neq 0$ since M is full rank, so $v^\top v = \sum_{i=1}^{n} v[i]^2 \neq 0$. This leads to contradiction. □

3 Intro to Fine-Grained Complexity

We will cover the Polynomial Method in the next few lectures.

1. The Polynomial Method: Computing Boolean functions with polynomials.
2. Applications: the fastest algorithm for the "Orthogonal Vectors (OV) Problem".

The motivation of the OV problem comes from fine-grained complexity. If we are able to prove that there is no truly sub-quadratic algorithm for the OV problem (the OV conjecture), then we will also show that there are no faster algorithms for many other problems in the class P.

Orthogonal Vectors (OV) Problem. Given as input $2n$ vectors of dimension d, $x_1, \cdots, x_n, y_1, \cdots, y_n \in \{0,1\}^d$, determine whether there are i, j such that $\langle x_i, y_i \rangle = 0$ over $\mathbb{Z}$. In other words, find i and j s.t. for all indices $k \in \{1, \cdots, d\}$, $x_i[k]$ and $y_j[k]$ are not both 1.

The fastest known algorithm for this problem runs in time

$$O\left(n^{2-1/O(\log(\frac{d}{\log n}))}\right).$$

For the setting where $d = c \log n$, this running time is

$$O\left(n^{2-1/O(\log c)}\right),$$

which is truly sub-quadratic if c is a fixed constant, but becomes quadratic as $c \to \infty$. It is conjectured that roughly quadratic time is required as $c \to \infty$:

Orthogonal Vectors Conjecture (OVC). For every $\varepsilon > 0$, OV in dimension $d = \omega(\log n)$ cannot be solved in time $O(2^{n-\epsilon})$.

The OV conjecture is closely related to an important hypothesis about the k-SAT problem.

k-SAT. The k-SAT problem is NP-complete for all $k \geq 3$. The best known running time for k-SAT on n variables is

$$O\left(2^{n\cdot(1-c_k)}\right)$$

for a constant $c_k > 0$ depending on k, but such that $c_k \to 0$ as $k \to \infty$. It is also conjectured that this is necessary:

Strong Exponential Time Hypothesis (SETH). For every $\varepsilon > 0$, there is a $k \geq 3$ such that k-SAT cannot be solved in time $O(2^{n(1-\varepsilon)})$.

It is known that SETH implies OVC. We will prove this in future lectures. In particular, faster algorithms for OV can be used to get faster algorithms for k-SAT.