CS 270 Combinatorial Algorithms and Data Structures, Spring 2015

Lecture 7: Polynomial identity testing

We have seen that simple algebraic ideas used in hashing have various applications.

Today we will see other simple algebraic ideas that are useful in designing fast (parallel) algorithms, and next time we will apply these ideas to the design of network coding.

String Equality

Suppose a company maintains multiple copies of the same huge data set.

From time to time they want to check that the copies are still the same.

Think of a data set is an n-bit string, for some very large n.

A naive way is to transmit all the n-bits from one server to another server.

Actually, one can prove that no deterministic algorithms could do better than that.

But a randomized algorithm could do much better, in $O(\log n)$ bits!

Think of the n-bits in server A to be $a_1 a_2 \ldots a_n$ and in server B to be $b_1 b_2 \ldots b_n$.

One idea is to consider the polynomials $A(x) = \sum_{i=1}^{n} a_i x_i$ and $B(x) = \sum_{i=1}^{n} b_i x_i$.

Choose a large enough finite field F (e.g. modular arithmetic over prime p).

Pick a random element $r \in F$, evaluate $A(r)$, send r and $A(r)$ to server B.

Server B checks whether $A(r) = B(r)$. If so, then "consistent"; otherwise "inconsistent".

That's the algorithm. How to analyze the success probability?

If the two strings are the same, then it will always be "consistent". So, if the algorithm says "inconsistent", it will never make a mistake (assuming no transmission error).

But the algorithm may make mistake when the two strings are not the same and it says "consistent". We would like to upper bound this error probability.

If the two strings are not the same, then $A(x) \not\equiv B(x)$ but we have chosen r s.t. $A(r) = B(r)$.

What is this error probability? It only happens when r is a root of $(A-B)(x)$.

Since $A(x)$ and $B(x)$ are polynomials of degree n, so is $(A-B)(x)$.

There are at most n roots of a degree n polynomial.

So, the error probability is at most $n/|F|$.

If we choose $|F| = 1000n$, then this is at most 0.001.

And the algorithm only needs to send $O(\log n)$ bits.

Note that if we would like to amplify the success probability, it is much more efficient to enlarge the field size rather than to repeat the algorithm multiple times.

Polynomial identity testing

We are given a multivariate polynomial $P(x_1, \ldots, x_n)$ and the objective is to determine if $P(x_1, x_2, \ldots, x_n) \equiv 0$, i.e. the polynomial is symbolically equal to the zero polynomial.

Of course, if the polynomial is given explicitly (e.g. $P(x_1, x_2, x_3) = x_1 x_2^2 x_3^3 + 4x_1^6 x_3 + x_2 x_3^2$), then this problem is straightforward, but the polynomial can also be given compactly (e.g. $P(x_1, x_2, x_3) = (x_1 - x_2^2)(x_3^3 + x_2^4)\ldots(x_1^2 + x_3)$, or $P(x_1, x_2, x_3) = \det\begin{pmatrix} x_1 & x_3 & x_2 + x_4^2 \\ 0 & x_2 - x_3 & 1 \\ x_2^3 & x_1 & x_4^2 x_2 x_3 \end{pmatrix}$) and the problem becomes difficult.

In general, there is no known deterministic polynomial time algorithm for this problem.

On the other hand, there is a simple randomized algorithm for this problem, by generalizing the idea that there are not many roots in a low degree polynomial.

In the following, when we talk about finite fields, you can just think of modular arithmetic over prime.

Lemma (Schwartz-Zippel Lemma) Let $Q(x_1, x_2, \ldots, x_n) \in F[x_1, x_2 \ldots x_n]$ be a multivariate polynomial of total degree d. Fix any finite set $S \subseteq F$, and let $r_1, r_2, \ldots, r_n$ be chosen independently and uniformly at random from S. Then $\Pr[Q(r_1, \ldots, r_n) = 0 \mid Q(x_1, \ldots, x_n) \not\equiv 0] \leq d/|S|$.

(the degree of any term is the sum of the exponents of the variables, and total degree is the max. degree)

Proof by induction. The base case $n = 1$ (single variable polynomial) is discussed before.

In the proof, we only focus on the case when $Q(x_1, \ldots, x_n) \not\equiv 0$.

Group the terms of $Q(x_1, \ldots, x_n)$ based on the variable x_1 to obtain

$$Q(x_1, \ldots, x_n) = \sum_{i=0}^{k} x_1^i \, Q_i(x_2, \ldots, x_n),$$

where k is the largest exponent of x_1 in Q, and each $Q_i(x_2, \ldots, x_n) \not\equiv 0$.

We condition on the event that $x_2 = r_2, x_3 = r_3, \ldots, x_n = r_n$.

Since $Q_k(x_2,\ldots,x_n) \not\equiv 0$ and it has only $n-1$ variables, $Q_k(r_2,\ldots,r_n) = 0$ with probability at most $\frac{d-k}{|S|}$ since the total degree of $Q_k(x_2,\ldots,x_n)$ is at most $d-k$.

Now, assuming $Q_k(r_2,\ldots,r_n) \neq 0$, then $g(x_1) = Q(x_1, r_2, \ldots, r_n)$ is a non-zero degree k single variable polynomial.

So, by the base case, $g(r_1) = 0$ with probability at most $k/|S|$.

Combining, we have $\Pr(Q(r_1,\ldots r_n) = 0)$

$$= \Pr(Q=0 \mid Q_k=0)\Pr(Q_k=0) + \Pr(Q=0 \mid Q_k\neq 0)\Pr(Q_k\neq 0)$$

$$\leq 1\cdot\left(\frac{d-k}{|S|}\right) + \left(\frac{k}{|S|}\right)\cdot 1 = \frac{d}{|S|}. \quad \square$$

Determinant testing

To check whether the determinant of an $n \times n$ matrix is identically equal to zero, we can pick a large enough prime field of size $\Theta(\text{poly}(n))$. This will ensure that with high probability that the determinant is still nonzero in this field (if originally it is nonzero). Then, we can just substitute random field elements in the variables and compute the numerical value of the determinant in polynomial time. By Schwarz-Zippel, if the determinant is nonzero, then the (success) probability that the numerical value is nonzero is very high.

Graph Matching

Given a bipartite graph $G_1 = (U, V; E)$ with $U = \{u_1, u_2, \ldots, u_n\}$ and $V = \{v_1, v_2, \ldots, v_n\}$, we would like to determine if there is a perfect matching in G_1, i.e. n node-disjoint edges.

Theorem (Edmonds) Let A be the $n \times n$ matrix obtained from $G_1 = (U, V; E)$ as follows:

$$A_{ij} = \begin{cases} x_{ij} & \text{if } u_i v_j \in E \\ 0 & \text{otherwise.} \end{cases}$$

Then G_1 has a perfect matching if and only if $\det(A) \not\equiv 0$.

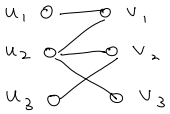
Proof: Recall that $\det(A) = \sum_{\text{permutation } \pi} \text{sign}(\pi)\, A_{1,\pi(1)} A_{2,\pi(2)} \cdots A_{n,\pi(n)}$.

Each term in the summation corresponds to a possible perfect matching of the graph.

If G_1 has no perfect matching, then there is a zero in each term in the summation, and hence $\det(A) \equiv 0$.

On the other hand, if there is a perfect matching, then the corresponding term is non-zero. Since no two terms have the same set of variables, and thus it won't be canceled out, and so $\det(A) \neq 0$. □

Pictorically,



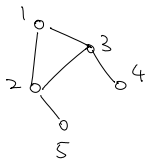
$$\begin{array}{c} \\ u_1 \\ u_2 \\ u_3 \end{array} \begin{array}{c} \begin{array}{ccc} v_1 & v_2 & v_3 \end{array} \\ \begin{bmatrix} x_{11} & 0 & 0 \\ x_{21} & x_{22} & x_{23} \\ 0 & x_{32} & 0 \end{bmatrix} \end{array} \qquad \begin{array}{c} \\ u_1 \\ u_2 \\ u_3 \end{array} \begin{array}{c} \begin{array}{ccc} v_1 & v_2 & v_3 \end{array} \\ \begin{bmatrix} 17 & 0 & 0 \\ 33 & 22 & 18 \\ 0 & 61 & 0 \end{bmatrix} \end{array}$$

Using the Edmonds' theorem, we can obtain a "simple" randomized algorithm to check whether a graph has a perfect matching. Just pick a (prime) field of size $\geq 2n$, substitute random value into the variables and check whether the determinant is zero. The most difficult step is to compute the determinant, which can be done in $O(n^3)$ time by Gaussian elimination. There is also an $O(n^w)$ time algorithm for computing the determinant where $w \approx 2.376$. So, theoretically, this is faster than combinatorial methods when the graph is dense.

There is also an algebraic formulation for matching in general graphs.

Given a graph $G=(V,E)$, we consider the following $|V| \times |V|$ matrix A such that

for each edge $e=ij$, we have $A_{ij} = x_e$ and $A_{ji} = -x_e$ and other entries to be zero.



$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & -x_{12} & x_{13} & 0 & 0 \\ x_{12} & 0 & x_{23} & 0 & -x_{25} \\ -x_{13} & -x_{23} & 0 & x_{34} & 0 \\ 0 & 0 & -x_{34} & 0 & 0 \\ 0 & x_{25} & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Theorem (Tutte) G has a perfect matching if and only if $\det(A) \neq 0$.

We won't give a formal proof here. The idea is that each term in the determinant is corresponding to a collection of vertex disjoint cycles whose union covers the whole vertex set. The key point is that those collections with odd cycles won't show up, because they come in pairs (by reversing an odd cycle) and they cancel each other (because of the skew-symmetry of the (signed) matrix). So, all terms in the determinant correspond to even cycle cover, and they can be decomposed into two perfect matchings, and one can check that they won't cancel out each other...

Fast algorithms: We briefly mention how to find a perfect matching quickly using this approach.

For simplicity, we only consider the bipartite matching problem.

A straightforward way is to check for each edge ij, whether $G-\{i,j\}$ has a perfect matching or not: if yes, we found one edge and recurse; if not, we delete ij and try another edge.

It will take $O(m \cdot n^\omega)$ time, where $\omega \approx 2.373$ is the matrix multiplication exponent, and this is too slow.

The next observation is that we can read off whether an edge is in a perfect matching by looking at the inverse. This is because $(A^{-1})_{i,j} = C_{j,i}/\det(A)$ where $C_{j,i}$ is the (j,i)-th cofactor of A.

Note that $C_{j,i} = \pm \det(A_{V-j, V-i})$ which is nonzero iff there is a perfect matching in $G-\{i,j\}$.

So, an edge ij is in a perfect matching iff $(A^{-1})_{i,j} \neq 0$.

This leads to an $O(n^{\omega+1})$-time algorithm for finding a perfect matching, by identifying an edge, updating the matrix and recomputing the inverse again (so as to identify another edge, so on).

The next observation is that we don't need to recompute the inverse in $O(n^\omega)$ time, because it is just a "rank-one update" in the matrix when we fix an edge to the matching, and its inverse can be "updated" in $O(n^2)$ time by the Sherman-Morrison-Woodbury formula.

So, this gives us an $O(n^3)$ algorithm.

Mucha-Sankowski and later Harvey showed that one can find a perfect matching in $O(n^\omega)$ time, by using a divide and conquer method in applying the Sherman-Morrison-Woodbury formula.

See the theses of Piotr Sankowski and Nick Harvey for various applications of this algebraic approach.

Parallel algorithm and isolation lemma

Another advantage to have this algebraic formulation is to have efficient parallel algorithms.

The main reason is that determinant can be computed in parallel efficiently.

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

Using this theorem, there is an efficient parallel algorithm to determine whether G has a perfect matching or not. Suppose there is a unique perfect matching, then we can construct the perfect matching in parallel as well (will be explained later).

Of course, there are graphs with (exponentially) many perfect matchings.

The difficulty in designing parallel algorithm is to coordinate the processors to search for the same matching.

The isolation lemma provides a surprising and elegant way to resolve this problem.

Lemma (Isolation lemma) Given a set system on a ground set of m elements, if we assign each element a weight independently and uniformly at random from $\{1, 2, \ldots, 2m\}$, then $\Pr[\text{there is a unique minimum weight set}] \geq \frac{1}{2}$.

(Remark: This is quite counterintuitive. Suppose the set system has 2^m sets. Then one may think that there are $2^m/(2m^2)$ sets of a given weight (since the max. weight $\leq 2m^2$).)

Proof: Let $\mathcal{F}$ be the set system.

Let v be an arbitrary element in the ground set.

Let $\mathcal{F}_{\bar{v}}$ be the set of subsets which do not contain v.

Let $\mathcal{F}_v$ be the set of subsets which contain v.

Consider $\alpha_v = \min_{F : \mathcal{F}_{\bar{v}}} w(F) - \min_{F : \mathcal{F}_v} w(F - v)$

Note that if α_v is negative, then v won't belong to a minimum weight set.

More precisely, if $w(v) > \alpha_v$, then v won't belong to a minimum weight set.

On the other hand, if $w(v) < \alpha_v$, then every minimum weight set must contain v.

We say an element v is <u>ambiguous</u> if $w(v) = \alpha_v$.

Note that α_v is independent of $w(v)$ and $w(v)$ is chosen uniformly from $\{1, \ldots, 2m\}$.

The probability that v is ambiguous is at most $1/2m$.

Therefore, by the union bound, some element is ambiguous is at most $1/2$.

Finally, observe that if two sets A and B have the minimum weight, then any element in $A - B$ and $B - A$ must be ambiguous.

Therefore, the probability that there are two sets with minimum weight is at most $1/2$. □

For the bipartite matching problem, we assign a random weight from $\{1, \ldots, 2m\}$

for each edge. By the isolation lemma, there is a unique minimum weight perfect matching with probability $1/2$.

So, now, our goal is to design a parallel algorithm to find the unique minimum weight matching.

For each edge i, we assign the value 2^{w_i} for the variable x_i.

Let the weight of the unique minimum weight perfect matching be W.

Then, only one matching will have weight 2^W; other matchings will have weight $2^{W'}$ which is 2^W times an even number.

Therefore, $\det(A)/2^k = \begin{cases} \text{even number if } k < W \\ \text{odd number if } k = W \\ \text{not divisible if } k > W \end{cases}$

We can find this k in parallel, after computing $\det(A)$ in parallel.

To construct the matching, we do the following for each edge $e=(u,v)$.

Construct G_{uv} which is obtained from G by deleting vertex u and vertex v.

Note that the weight of a minimum weight matching of G_{uv} + the weight of an edge uv
= the weight of a minimum weight matching containing the edge uv.

Let M be the unique minimum weight matching in G.

Then $uv \in M$ if and only if the weight of a minimum weight perfect matching in G_{uv} is equal to $W - w_{uv}$.

Therefore, to obtain a parallel algorithm, we can construct G_{uv} in parallel and then add an edge uv in the matching iff $\det(A_{uv}) = 2^{W - w_{uv}}$.

So the whole process can be run in parallel "efficiently".

In general, other algebraic algorithms can also be made parallel.

References: Material in this lecture is from chapter 7 of "Randomized Algorithms" by Motwani & Raghavan.

Open question: Given a graph where each edge is colored red or blue, we would like to determine if there is a perfect matching with exactly k red edges. It is an exercise to show that there is a randomized polynomial time algorithm for this problem using the algebraic approach, but

it has been an open question whether there is a deterministic algorithm for the problem.