CSC 5450 Randomness and Computation

Week 8: Algebraic Techniques

Plan: ① Polynomial identity testing, Schwarz-Zippel lemma

② Matching, isolation lemma, parallel algorithms

③ Network coding

④ Matrix rank, independent columns

Questionnaire The result is posted in the homework page.

The most popular topics are ① Graph algorithms ② Approximation algorithms ③ Complexity ④ Web graphs. (in descending order)

I will be totally democratic and follow this order from week 11 to week 14.

Project Outline while I may not cover your favorite topic in the course, it may be even better in the sense that you can do it in your course project.

Please submit your project outline by Apr 2 (email) with the following content:

① Project title ② Abstract ③ an one page outline ④ references

This will be counted 10% of the course grade.

The purpose is to have a clear idea of what you are going to do in the project, so that I can give some feedback.

Remember that this project can be closely related to your research and could be a survey, but it should be mathematical and theoretical.

Please send me email if you have any questions.

Fingerprinting

Recall the string equality example we've seen in the first week.

To check whether two n-bit strings $x_1x_2\dots x_n$ and $y_1y_2\dots y_n$ are equal, we can just transmit $O(\log n)$ bits and the test is correct with high probability.

The idea is to think of each string as a degree n polynomial $P(z) = \sum_{i=1}^{n} x_i z^i$, and we do the arithmetic operations over a large enough finite field (e.g. pick a large enough prime q, and do

arithmetic modulo q), and just transmit $P(r)$ for the test, where r is a random field element.

Choose the field size to be $\Theta(\text{poly}(n))$, say $\Theta(n^4)$.

Then each field element can be represented in $\Theta(\log(n))$ bits, and each operation can be done in $\Theta(\log(n))$ time.

The algorithm makes mistakes only if the two polynomials P, Q are not equal but $P(r) = Q(r) \Leftrightarrow (P-Q)(r) = 0$.

Since P, Q are of degree at most n, the polynomial $P-Q$ has at most n roots.

Since the field size is chosen to be $\Theta(n^4)$, the error probability is at most $O(1/n^3)$.

This technique is called "fingerprinting", which maps a long string into a short string that "preserves identity".

It is also useful in other problems, see the pattern matching example in MR 7.6 or L06 in 2011.

Today we generalize this idea to multivariate polynomial and see some interesting applications.

Polynomial identity testing (MR 7.2)

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 implicitly (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 $P(x_1, x_2, x_3) = \det\begin{pmatrix} x_1 & x_3 & x_2 + x_4^2 \\ 0 & x_2 - x_3 & 1 \\ x_2^3 & x_1 & x_4^3 x_2 x_3 \end{pmatrix}$) 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 [illegible] x_1 [illegible] and each $Q_i(x_2, \ldots, x_n) \not\equiv 0$

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 $Q_k(r_2,\ldots,r_n) \not\equiv 0$, 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 $\Pr(Q(r_1,\ldots r_n) = 0)$

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

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

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.

This determinant test will be useful in all the applications later.

Bipartite Matching (MR 7.3)

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

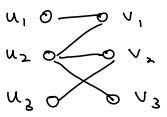
Each term in the summation corresponds to a possible perfect matching of the graph.

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.

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 $\det(A) \not\equiv 0$. □

Pictorically,



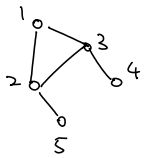
$$\begin{array}{c} \\ u_1 \\ u_2 \\ u_3 \end{array}\begin{array}{c} \begin{array}{ccc} v_1 & v_2 & v_3 \end{array} \\ \begin{bmatrix} x_{11} & 0 & 0 \\ x_{21} & x_{22} & x_{23} \\ 0 & x_{32} & 0 \end{bmatrix} \end{array} \qquad \begin{array}{c} \\ u_1 \\ u_2 \\ u_3 \end{array}\begin{array}{c} \begin{array}{ccc}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

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 $\det(A) \not\equiv 0$.

We won't give a proof here. See [1] for a proof and algebraic formulations for some other problems.

One may also want to find a perfect matching (instead of just knowing whether it exists). A straightforward algorithm is to check for each edge whether $G-uv$ has a perfect matching (if yes, delete it; if no, keep it), but this takes $O(m \cdot n^w)$ time. It is possible to find a perfect matching in $O(n^w)$ time as well; see [1] and the references therein.

Parallel algorithm and isolation lemma (MR 12.4)

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 (Thm 12.9 of MR)

Thm (Thm 12.9 of MR)

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 very 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 $A - B$ must be ambiguous (verify).
Therefore, the probability that there are two sets with minimum weight is at most $1/2$. □

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_1 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_1.

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 $\det(A_{uv}) = 2^{W-w_{uv}}$.

(There is a smarter way to compute $\det(A_{uv})$ for all uv by computing the adjoint matrix of A, but the above method is enough to get a polylog time algorithm with a polynomial number of processors.)

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

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

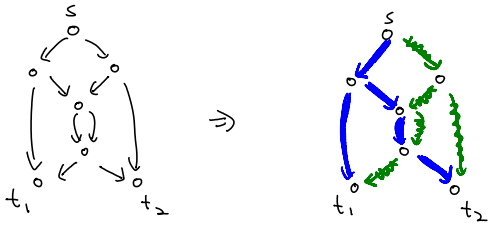
Network Coding [2]

Given a directed acyclic graph, a source node s and a set of receiver nodes

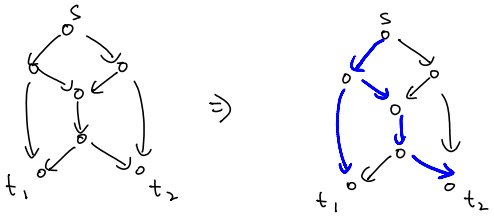
Given a directed acyclic graph, a source node s and a set of receiver nodes $\{t_1, t_2, \ldots, t_m\}$, the <u>multicasting</u> problem is for the source to send data to all the receivers, while the objective is to maximize the rate (speed) of transmission.

In traditional computer network setting, each intermediate node can be used to store and forward any data. Assume that each edge can transmit one unit of data per step. To increase the transmission rate, we need to find edge-disjoint trees connecting the source to all receivers.

In this network, we can find two edge disjoint trees connecting the source to all receivers.

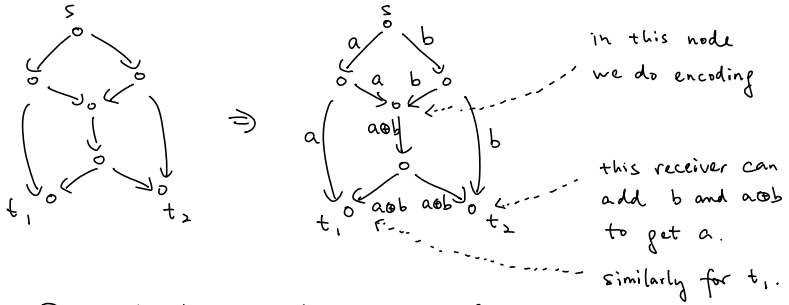


But in this example only one edge-disjoint tree can be found.



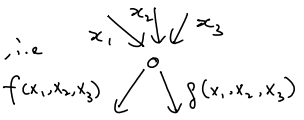
The idea of network coding is to do <u>encoding</u> in intermediate nodes so that the receivers can <u>decode</u> the messages.

using coding the sender can send two units of data to the receivers



More generally network coding allows ① arithmetic operations over a field

② the data on an edge is a general function of its predecessors, i.e



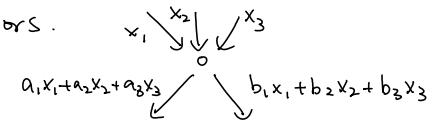
Surprisingly the rate of multicasting could be significantly improved if network coding is used. In fact it can achieve optimal rate for multicasting.

<u>Theorem</u> ([3] Ahlswede, Cai, Li, Yeung) Whenever the source has k edge-disjoint paths from the source to every receiver, then the source can transmit k units of data simultaneously to all receivers.

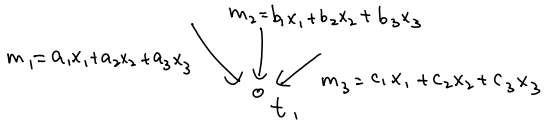
(The rate when network coding is used and the rate when network coding is not used could be unbounded. Also, this is optimal.)

An important algorithmic question is how to find an efficient encoding and decoding scheme.

An important algorithmic question is how to find an efficient encoding and decoding scheme. It is proved that <u>linear</u> network coding is enough to achieve the optimal rate for multicasting. By linear network coding, we mean the data on an edge is a linear combination of its predecessors.



A receiver can decode if what it receives are linearly independent



$\Rightarrow$

$$m_1 = a_1x_1 + a_2x_2 + a_3x_3$$
$$m_2 = b_1x_1 + b_2x_2 + b_3x_3$$
$$m_3 = c_1x_1 + c_2x_2 + c_3x_3$$

$\Rightarrow \vec{m} = A\vec{x}$

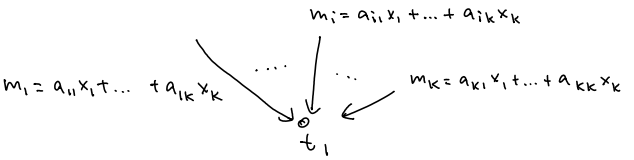
so $\vec{x}$ can be recovered if A is of full-rank.

Later, deterministic polynomial time algorithms are obtained to find an optimal encoding scheme for multicasting, i.e. the coefficients of the linear combinations.

Here, we present a randomized polynomial time algorithm to find an optimal encoding scheme for multicasting. It has the advantages of being very simple and totally decentralized, which is a key for the algorithm to be practical.

The algorithm is very simple: just assign a random coefficient from a large enough field to every pair of edges $(\vec{uv}, \vec{vw})$. More precisely, suppose the source has k edge-disjoint paths to every receiver and want to transmit k messages $x_1, x_2, \ldots, x_k$ to all receivers simultaneously. Without loss of generality we assume the source has k outgoing edges $a_1, \ldots, a_k$. Then the source sends the i-th message (i.e x_i) on the i-th outgoing edge a_i. Then we follow a topological ordering to process the intermediate nodes. For each intermediate node, each outgoing edge is an independent random linear combination of its incoming edges. For each receiver, it solves a system of linear equation to recover $x_1, x_2, \ldots, x_k$.

Without loss of generality, we assume that each receive has exactly k incoming edges. To achieve optimal rate, we need to prove that the equations have a unique solution. Equivalently, the corresponding matrix is of full-rank. See the following figure.



$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & a_{2k} \\ \vdots & & \ddots & \vdots \\ a_{k1} & a_{k2} & \cdots & a_{kk} \end{bmatrix}$$

the receiver can decode if this matrix is of full-rank

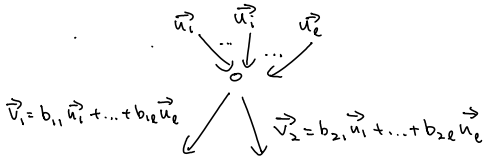
t_1 $[a_{K1}\ a_{K2} \dots a_{KK}]$

Each edge carries a linear combination of the original messages.

Suppose the linear combination is $a_1x_1 + \dots + a_Kx_K$.

Then we say $(a_1, a_2, \dots, a_K)$ is a <u>global encoding vector</u> for the edge.

The global encoding vector is a random linear combination of its predecessors.



We call the coefficients b_{ij} are the <u>local encoding coefficients</u>.

Once the local coefficients are fixed, then all the global encoding vectors are fixed.

So our task is to choose the local encoding coefficients.

Think of each local coefficient is a variable. We argue that when the i-th node is processed (according to the topological ordering), each entry in the global encoding of its outgoing edges is a multivariate polynomial of the local encoding coefficients with total degree at most i.

We can prove this by induction. For i=1 this is easy to see. Assuming this for i≤t it is also easy to see for i=t+1, since the total degree increases by at most one at each intermediate node.

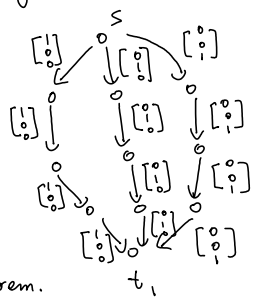
Therefore, each entry in the receiver matrix is just a degree n multivariate polynomial of the local encoding coefficients. To prove that the receiver matrix A is of full-rank, it is equivalent to proving that $\det(A) \neq 0$. Since A is a $k \times k$ matrix, $\det(A)$ is just a multivariate polynomial of the local encoding coefficients of total degree kn. If we can show that this polynomial is non-zero, then if we pick a field of size $\geq kn^3$ and assign random values to the local encoding coefficients then $\det(A) \neq 0$ with probability at most $1/n^2$. By the union bound every receiver matrix is of full rank with probability at least $1 - \frac{1}{n}$.

It remains to show that $\det(A) \neq 0$ for a receiver matrix A. To show this it suffices to demonstrate one assignment of the local encoding coefficients so that $\det(A) \neq 0$. To do this, consider a set of k edge-disjoint paths from the source to a

receiver t_1. Assign the local encoding coefficient of (uv, vw) to be one if uv, vw is on some path in the k edge-disjoint paths; otherwise assign zero.

Then it can be seen that the receiver matrix is an identity matrix and thus the receiver matrix is not identically equal to zero.

See the figure for an illustration



This concludes that $\det(A)$ is a "low"-degree (non-zero) polynomial of the local encoding coefficients, and thus concludes the theorem.

Efficient Encoding [4]

Consider the problem of implementing the encoding operation at a node v.

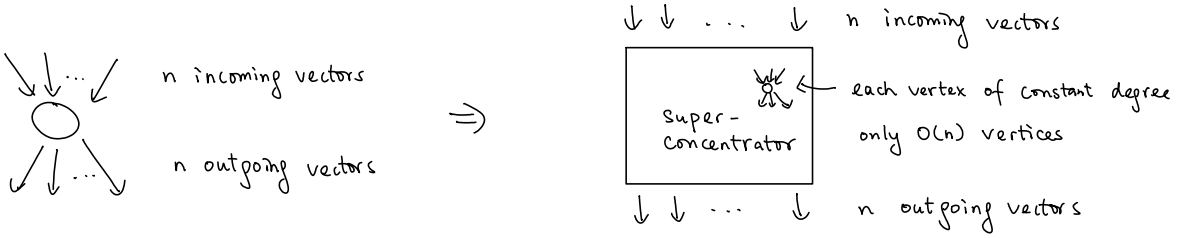
Suppose each vector is of d-dimensional. There are n incoming vector and n outgoing vector.

Then each outgoing vector can be computed in $O(dn)$ time, and all outgoing vectors can be computed in $O(dn^2)$ time.

Can we do it faster? Observe that the encoding can be done quickly if indegree is small.

So, it would be good if there is a transformation to reduce the degree while preserving connectivity.

A superconcentrator comes to mind. (Recall lecture 4)



An important point is that any k inputs can connect to any k outputs by vertex disjoint paths.

Thus, the connectivity from the source to a receiver would not decrease.

Therefore, we can first apply the transformation before doing network coding.

After the transformation, each vertex in the resulting graph has constant indegree.

So, the encoding in each vertex (all the outgoing edges) can be done in $O(d)$ time.

All the vertices in a superconcentrator can be done in $O(d \cdot n)$ time, as a vertex with indegree n is replaced by a superconcentrator with n inputs, and it has only $O(n)$ vertices.

Hence, the vectors of the outgoing edges of a vertex in the original graph can be computed in $O(dn)$ time, faster than the straightforward $O(dn^2)$ time algorithm.

Note that this is optimal, since only writing down n output vectors requires $\Omega(dn)$ time.

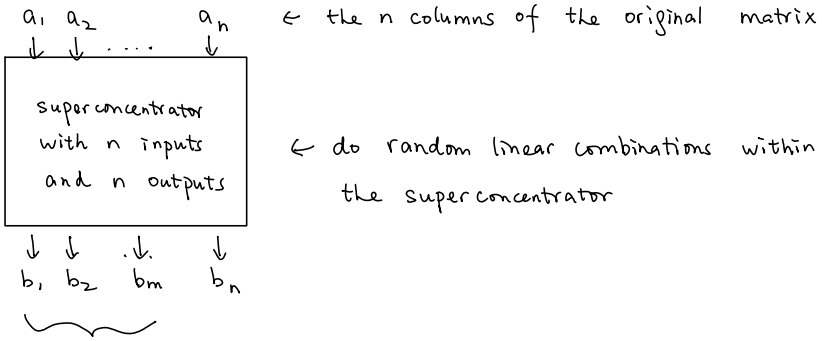
Note that this is optimal, since only writing down n output vectors requires $\Omega(dn)$ time.

Matrix Rank [5]

Given an $m \times n$ matrix A for $m \leq n$, we consider the problem of finding a maximum set of linearly independent columns. This maximum number is called the rank of A, denoted by $\text{rank}(A)$.

Finding a set of $\text{rank}(A)$ linearly independent columns can be done in $O(n \cdot m \cdot r^{w-2})$ time, by doing Gaussian elimination with fast matrix multiplication.

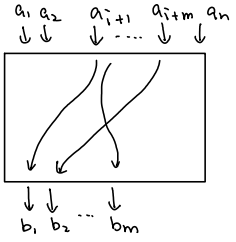
Interestingly, if we just want to compute $\text{rank}(A)$ (i.e. only the value), then we can use a superconcentrator to do the job.



Let $A = \begin{pmatrix} | & | & & | \\ a_1 & a_2 & \cdots & a_n \\ | & | & & | \end{pmatrix}$ and $B = \begin{pmatrix} | & | & & | \\ b_1 & b_2 & \cdots & b_m \\ | & | & & | \end{pmatrix}$. We claim that $\text{rank}(A) = \text{rank}(B)$ w.h.p.

The proof is the same as in network coding.

Say $a_{i+1}, \ldots, a_{i+m}$ are a maximum set of linearly independent columns in A.



By the property of the superconcentrator, there are disjoint paths connecting them to $b_1, \ldots, b_m$.

If we set the coefficients along the paths to be one and all other "local encoding coefficients" to be zero (just like the network coding proof), then $B = \begin{pmatrix} | & | & & | \\ a_{i+1} & a_{i+2} & \cdots & a_{i+m} \\ | & | & & | \end{pmatrix}$ up to permuting rows, and hence $\text{rank}(A) = \text{rank}(B)$ with <u>non-zero</u> probability.

Since B is of full rank in that situation, it implies that $\det(B) \neq 0$ with non-zero probability.

So, $\det(B)$ is a <u>non-zero</u> polynomial.

As in the network coding proof, if we think of each random coefficient as a variable, then $\det(B)$ is just a degree $O(n)$ polynomial, since there are $O(n)$ nodes in the superconcentrator.

So, if the field size is large enough, say $\Theta(n^3)$, then $\text{rank}(A) = \text{rank}(B)$ with high probability.

by the Schwarz-Zippel lemma.

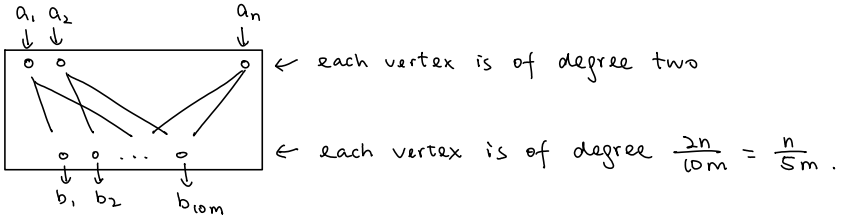
(If the field size is not large enough, we can use an extension field to enlarge the field.)

What is the running time? As in the analysis in efficient encoding, the vectors $b_1, \ldots, b_m$ can be computed in $O(mn)$ time. Since B is an $m \times m$ matrix, rank(B) can be computed in $O(m^\omega)$ time. So, the total running time to compute rank(A) is $O(mn + m^\omega)$, which is faster than $O(n \cdot m^{\omega-1})$ by Gaussian elimination.

Finding independent columns [5]

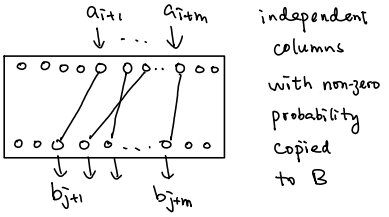
We show a even faster algorithm to compute rank(A), and use it to also find independent columns.

The idea is to use a magical graph. (recall lecture 4) to do the random linear combinations.



By a simple probabilistic argument (similar to lecture 4), we can efficiently construct (randomly) a magical graph with degree two vertices on top and degree $n/5m$ vertices at bottom, with the magical property that any subset of m vertices on top has a perfect matching to the bottom.

By the same argument as in the above proof using superconcentrator, the perfect matching (plays the same role as the disjoint paths) ensures that with <u>non-zero</u> probability rank(B) = rank(A) (by setting the coefficients in the matching to be one and other coefficients zero).



Thus, as before, rank(A) = rank(B) w.h.p. by the Schwarz-Zippel lemma.

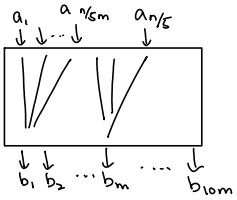
What is the running time? Each column in A is involved in two columns in B.

Let $|A|$ be the number of nonzero entries in A. Then B can be computed in $O(|A|)$ time.

So, rank(A) can be computed in $O(|A| + m^\omega)$ time, faster than $O(nm + m^\omega)$ by superconcentrator when A is sparse.

Finding Columns

To find the independent columns in A, we first find a maximum set of independent columns in B.



each column of B is a linear combination of $\frac{n}{5m}$ columns in A, and so a set of at most m independent columns in B correspond to at most $m \cdot (\frac{n}{5m}) = \frac{n}{5}$ columns in A.

These $n/5$ columns in A is of the same rank as B.

Therefore, we can delete the other columns in A, so deleting at least $4n/5$ columns.

So, in $O(|A| + m^\omega)$ time, we delete a constant fraction of columns in A.

In $O(\log n)$ iterations, we reduce the number of columns in A to $O(m)$, and then we can solve the problem directly by Gaussian elimination in $O(m^\omega)$ time.

Thus, the overall complexity is $O((|A| + m^\omega) \log n)$ time, which could be considerably faster than direct Gaussian elimination in $O(n \cdot m^{\omega-1})$ time.

Application: Since computing matrix rank can be used to solve combinatorial optimization problems, this has some combinatorial applications, e.g. finding a small matching faster (e.g. finding a maximum matching in a very unbalanced bipartite graph).

References

[1] Cheung. Algebraic algorithms in combinatorial optimization. M.Phil. CUHK.

[2] Fragouli, Soljanin. Network coding fundamentals

[3] Ahlswede, Cai, Li, Yeung. Network information flow.

[4] Cheung, Lau, Leung. Graph connectivities, network coding, and expander graphs.

[5] Cheung, Kwok, Lau. Fast matrix rank algorithms and applications.