CS 270 Combinatorial Algorithms and Data Structures, Spring 2015

Lecture 8 : Network coding

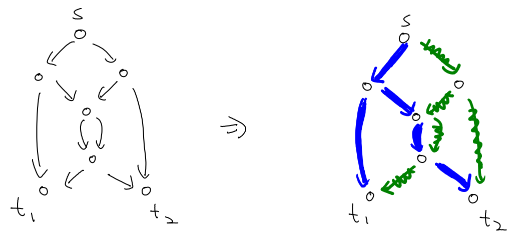
Network coding is a new method to transmit data in a computer network, and randomization helps make it work in a fully distributed manner. Interestingly, these ideas also lead to fast algorithms for computing matrix rank.

Network Coding

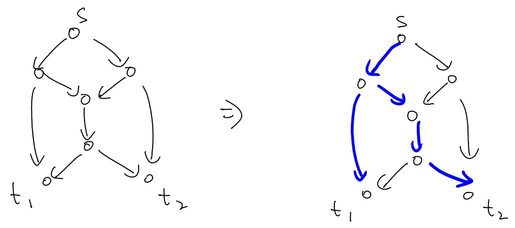
Given a directed acyclic graph, a source node s and a set of receiver nodes $\{t_1, t_2, \dots, t_m\}$, the multicasting problem is for the source to send data to all the receivers, while the objective is to maximize the rate (speed) of transmission.

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.

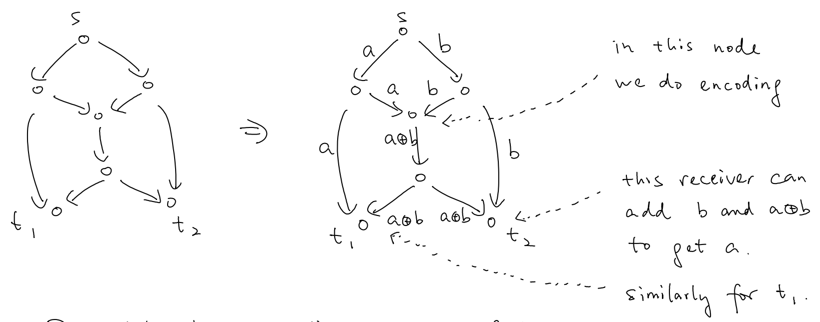


But in this example only one edge-disjoint tree can be found.



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

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



s
t1
t2
⇒
s
a
b
a
b
a
a⊕b
b
a⊕b
a⊕b
t1
t2
in this node we do encoding
this receiver can add b and a⊕b to get a.
similarly for t1.

More generally network coding allows ① arithmetic operations over a field similarly for t_1.

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

$$\begin{tikzcd}
x_1 \arrow[dr] & x_2 \arrow[d] & x_3 \arrow[dl] \\
& \circ \arrow[dl] \arrow[dr] & \\
f(x_1,x_2,x_3) & & g(x_1,x_2,x_3)
\end{tikzcd}$$

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> (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. 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.

$$\begin{tikzcd}
x_1 \arrow[dr] & x_2 \arrow[d] & x_3 \arrow[dl] \\
& \circ \arrow[dl] \arrow[dr] & \\
a_1x_1+a_2x_2+a_3x_3 & & b_1x_1+b_2x_2+b_3x_3
\end{tikzcd}$$

A receiver can decode if what it receives are linearly independent

$$\begin{tikzcd}
& m_2=b_1x_1+b_2x_2+b_3x_3 \arrow[d] & \\
m_1=a_1x_1+a_2x_2+a_3x_3 \arrow[r] & \underset{t_1}{\circ} & m_3=c_1x_1+c_2x_2+c_3x_3 \arrow[l]
\end{tikzcd}$$

$$\Rightarrow \quad \begin{aligned} m_1 &= a_1x_1+a_2x_2+a_3x_3 \\ m_2 &= b_1x_1+b_2x_2+b_3x_3 \\ m_3 &= c_1x_1+c_2x_2+c_3x_3 \end{aligned} \quad \Rightarrow \quad \vec{m}=A\vec{x}$$

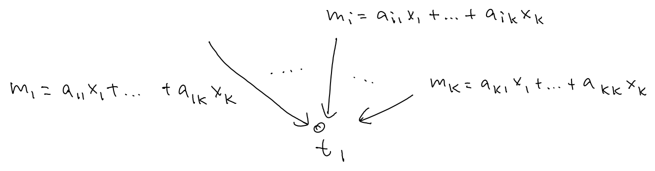
so $\vec{x}$ can be recovered if A is of full-rank.

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 $(\overrightarrow{uv}, \overrightarrow{vw})$. 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.


$m_i = a_{i1}x_1 + \ldots + a_{ik}x_k$
$m_1 = a_{11}x_1 + \ldots + a_{1k}x_k$
$m_k = a_{k1}x_1 + \ldots + a_{kk}x_k$
t_1

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

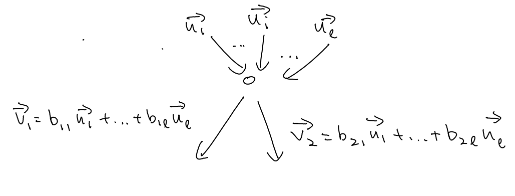
the receiver can decode if this matrix is of full-rank

Each edge carries a linear combination of the original messages.

Suppose the linear combination is $a_1x_1 + \ldots + a_kx_k$.

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

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


$\vec{u_1}$
$\vec{u_i}$
$\vec{u_e}$
$\vec{v_1} = b_{11}\vec{u_1} + \ldots + b_{1e}\vec{u_e}$
$\vec{v_2} = b_{21}\vec{u_1} + \ldots + b_{2e}\vec{u_e}$

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.

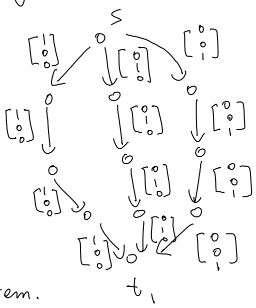
Think of each local coefficient is a variable. We argue that when the i-th node is processed (according to the topological ordering), each entry in the global encoding of its outgoing edges is a multivariate polynomial of the local encoding coefficients with total degree at most i.

We can prove this by induction. For $i=1$ this is easy to see. Assuming this for $i \le t$ 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 $\det(A) \not\equiv 0$ 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 $\overset{\text{non-zero}}{\vee}$ "low"-degree polynomial of the local encoding coefficients, and thus concludes the theorem.

Efficient Encoding

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

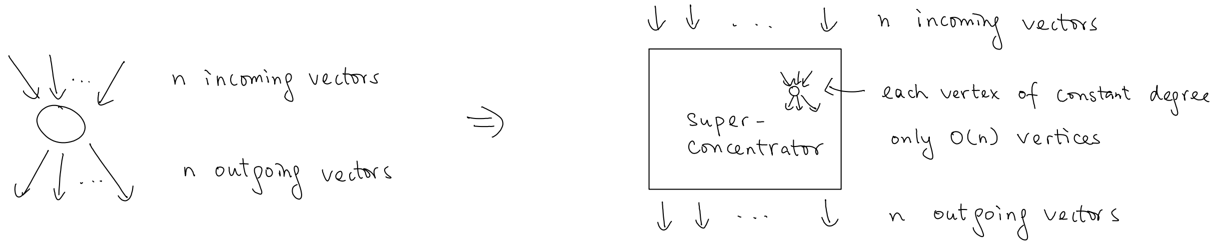
Suppose each vector is of d-dimensional. There are n incoming vector and n outgoing vector.

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 rescue (see next lecture for its construction)



A superconcentrator is a directed acyclic graph with n input nodes and n output nodes, with $O(n)$ internal nodes each with constant indegree and outdegree.

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

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.

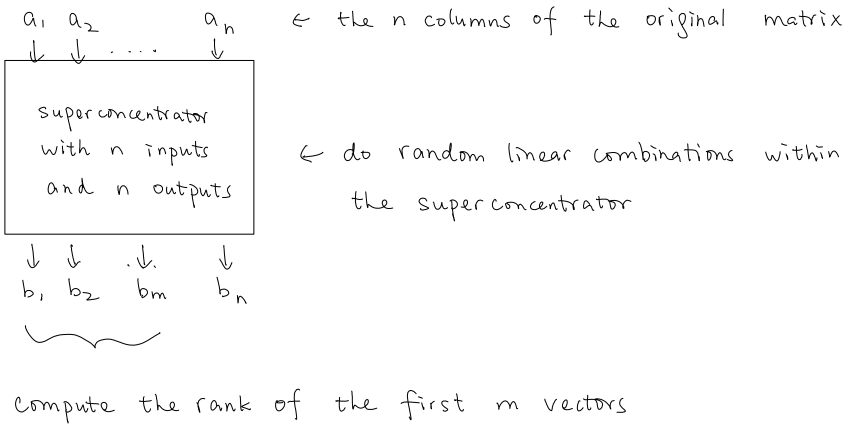
Note that this is optimal, since only writing down n output vectors requires $\Omega(dn)$ time.

Matrix Rank

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

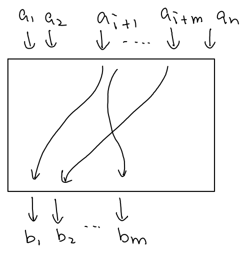
Finding a set of $\text{rank}(A)$ linearly independent columns can be done in $O(n \cdot m \cdot r^{w-2})$ time, by doing Gaussian elimination with fast matrix multiplication.

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



a_1 a_2 . . . a_n
← the n columns of the original matrix
superconcentrator with n inputs and n outputs
← do random linear combinations within the superconcentrator
b_1 b_2 b_m b_n
compute the rank of the first m vectors

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 rank(A) = 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 rank(A) = 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 rank(A) = rank(B) with high probability, by the Schwarz-Zippel lemma.

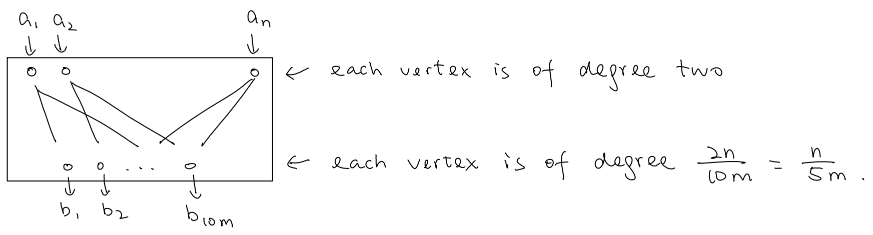
(If the field size is not large enough, we can use an extension field to enlarge the field.)

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

Finding independent columns

We show an even faster algorithm to compute $\text{rank}(A)$, and use it to also find independent columns.

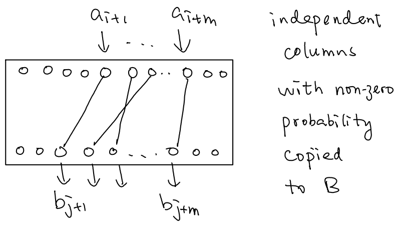
The idea is to use a magical graph. (see next time) to do the random linear combinations.



a_1 a_2 a_n
← each vertex is of degree two
← each vertex is of degree $\frac{2n}{10m} = \frac{n}{5m}$.
b_1 b_2 b_{10m}

By a simple probabilistic argument (see next time), 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 $\text{rank}(B) = \text{rank}(A)$ (by setting the coefficients in the matching to be one and other coefficients zero).



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

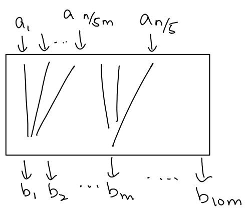
What is the running time? Each column in A is involved in two columns in B.

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

So, $\text{rank}(A)$ 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 and discussions There are several surveys about network coding. One could also take a look at the textbook "Information theory and network coding" by Yeung. The algorithm for matrix rank is in the paper "Fast matrix rank algorithms and applications".

There are more and more results using algebraic techniques to design fast (sometimes exponential) algorithms (e.g. faster Hamiltonian cycle, a recent paper "Shortest two disjoint paths in polynomial time").

There are two interesting open questions in algebraic algorithms: One is to design faster algorithms for weighted problems (e.g. weighted bipartite matching), as the current technique only gives $O(Wn^\omega)$-time algorithm where W is the maximum weight of an edge, and it would be great to significantly reduce the dependency on W. Another is to design an $O(n^\omega)$-time algorithm to find the maximum number of edge disjoint s-t paths, where the current best is an $O(m^\omega)$ time algorithm.

Polynomial identity testing is also very useful in designing efficient "interactive proofs". More generally, randomness is very useful in "checking" things efficiently. A prime example is the celebrated PCP (probabilistically checkable proof) theorem. It would be nice to have one lecture on these topics.