CS 761 : Randomized Algorithms , Spring 2018 , Waterloo

Lecture 9 : Network coding

Network coding is a novel method to transmit data in a computer network.

Randomization makes it work in a fully distributed manner.

Interestingly, these ideas can be used to design fast algorithms for computing matrix rank.

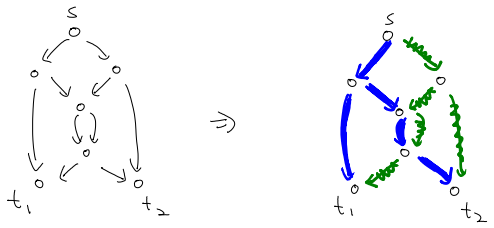
Network information flow

For simplicity, we assume the graph is a directed acyclic graph.

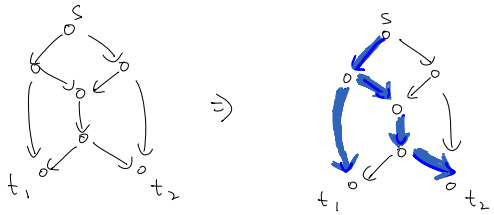
Multicasting : Given a directed acyclic graph $G=(V,E)$, a source vertex $s \in V$, and a set of receiver vertices $\{t_1, \ldots, t_\ell\} \subseteq V$, the multicasting problem is for the source to send data to all the receivers simultaneously, and the objective is to maximize the rate (speed) of transmission.

Tree packing : In traditional computer network setting, each intermediate node can be used to store and forward 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.



Two edge-disjoint trees in this example.



Only one edge-disjoint tree can be found, even though there are two edge-disjoint paths from s to each t_i.

It is a computational hard problem to compute the maximum number of such edge-disjoint trees.

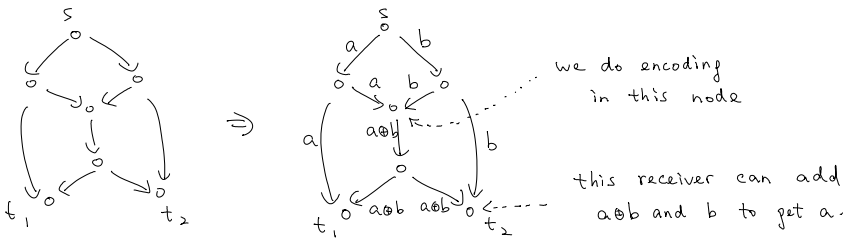
Also, there are graphs with high "connectivity" from the source to each receiver, but there are very few edge-disjoint trees connecting the source to all receivers.

That is, while it is easy for the source to send data quickly to each individual receiver, it becomes difficult for the source to send data quickly to all receivers simultaneously.

Network coding: The idea of network coding is that information flow is different from standard commodity flow.

We could do encoding in intermediate nodes so that the receivers can decode the messages and the throughput is increased!

can send two units of information in this example by doing encoding and decoding



More generally, network coding allows

① arithmetic operations over a larger field (instead of just bit-wise operations).

② the data sent on an edge is any function of its predecessors (not just XOR).

Surprisingly, the rate of transmission could be significantly improved if network coding is used.

In fact, it can always achieve the optimal rate for multicasting.

Theorem (max-information-flow min-cut theorem)

If the source has k edge-disjoint paths to each receiver, then the source can send k units of data to all receivers simultaneously with the use of network coding.

Remarks: ① This is clearly optimal. This is an amazing result that opens up a new research area.

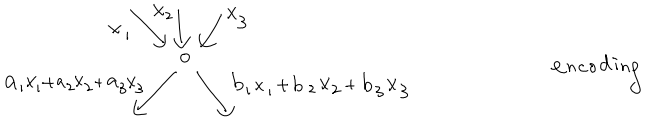
② There are examples for which the ratio of the rate achieved by network coding to the rate achieved without using network coding is unbounded.

Linear Network Coding

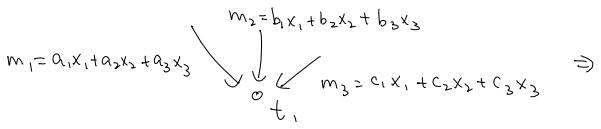
An important algorithmic question is to find an optimal encoding and decoding scheme efficiently.

The next major result is that linear network coding is enough to achieve the optimal rate for multicasting.

In linear network coding, the data sent on an edge is always a linear combination of its predecessors.



A receiver can decode if what it receives are linearly independent.



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

$\vec{x}$ can be uniquely determined if A is of full rank.

Deterministic algorithm: Subsequently, deterministic polynomial time algorithms are obtained to find an optimal coding scheme for multicasting, i.e. the coefficients of the linear combinations. However, these algorithms will be very difficult to be used in practice, as it requires a "central authority" to look at the whole graph and carefully choose the coefficients and tell everyone how to encode.

Randomization: We present a randomized polynomial time algorithm to find an optimal coding scheme for multicasting. The proof also provides an alternate proof for the max-information-flow min-cut theorem and the linear coding thm. Most importantly, the algorithm is very simple and totally decentralized, a key feature to be practical.

Setting: By a simple reduction, we can assume that the source has exactly k edges, each reciever has exactly k incoming edges, where k is the edge-connectivity from the source to the receivers. The source would like to send k messages $X_1, \dots, X_k$ to all receivers.

Algorithm: Pick a large enough (prime) field $\mathbb{F}$.

The Source sends $X_1, \dots, X_k$ on its outgoing edges, X_i on its i-th outgoing edge.

Follow a topological ordering $v_1 = s, v_2, \dots, v_n$ of the directed acyclic graph.

For vertex v_i, call the incoming messages $m_1, m_2, \dots, m_\ell$.

For each outgoing edge e of v_i, send a random linear combination of $m_1, \dots, m_\ell$,

i.e. send $b_1^{(e)} m_1 + \dots + b_\ell^{(e)} m_\ell$ on e, where each $b_i^{(e)}$ is an independent random element in $\mathbb{F}$.

Global encoding vector and local encoding coefficients: These are important definitions for the analysis. Each edge is sending a linear combination of the source messages, $c_1x_1 + c_2x_2 + \dots + c_kx_k$, which follows by induction on the topological ordering.

We call $(c_1, \dots, c_k)$ the global encoding vector on this edge, the actual linear combination that its sending.

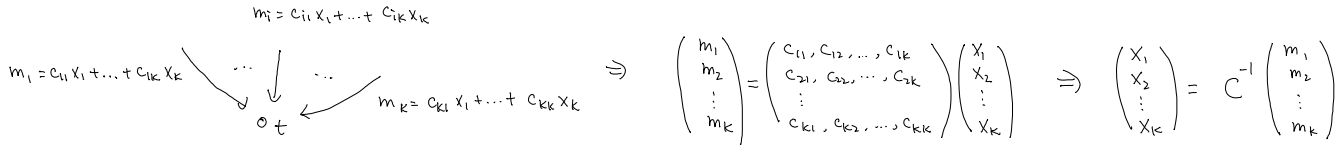
For each edge e, the (random) coefficients $b_1^{(e)}, b_2^{(e)}, \dots$ are called the local encoding coefficients.

The local encoding coefficients are the decisions that we need to make (which we decide to make randomly).

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

Decodability: For a receiver t with k incoming edges, let the global encoding vector of its i-th incoming edge be $(c_{i1}, c_{i2}, \dots, c_{ik})$ and the i-th incoming message is m_i.

Then, whether t can decode the original messages depends on whether the k incoming global encoding vectors are linearly independent, or equivalently whether the $k \times k$ matrix $C = \begin{pmatrix} c_{11}, c_{12}, \dots, c_{1k} \\ c_{21}, c_{22}, \dots, c_{2k} \\ \vdots \\ c_{k1}, c_{k2}, \dots, c_{kk} \end{pmatrix}$ is of full rank, or equivalently whether $\det(C) \neq 0$.



Analysis: We need to analyze the probability that $\det(C) \neq 0$ if we choose all local coefficients randomly.

Think of each local encoding coefficient as a variable (the decision variable).

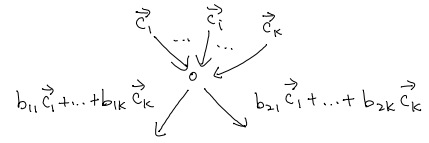
Each entry of a global encoding vector is a multivariate polynomial involving the local encoding coefficients.

Claim: For vertex v_i in the topological ordering, for each outgoing edge e of v_i, each entry c_j of its global encoding vector is a multivariate polynomial of the local encoding coefficients of total degree at most $i-1$.

Proof: This follows from a simple induction.

The base case is clearly true, with vector $(0, 0, \dots, \underset{\text{i-th position}}{1}, 0, \dots, 0)$ on the i-th outgoing edge of s.

In each step, we just add degree one to the polynomial, as shown in the picture.



□

By the claim, each entry in a receiver matrix C is just a multivariate polynomial of the local encoding coefficients of total degree at most n.

This implies that $\det(C)$ is a multivariate polynomial of the local coefficients of total degree at most kn.

If $\det(C) \not\equiv 0$, then by the Schwartz-Zippel lemma, the probability that $\det(C) = 0$ when we put in random values is at most $\frac{kn}{|\mathbb{F}|}$.

By choosing $|\mathbb{F}| \geq kn^3$, this probability is at most $\frac{1}{n^2}$ for one receiver.

By the union bound, every receiver is of full rank with probability at least $1 - \frac{1}{n}$.

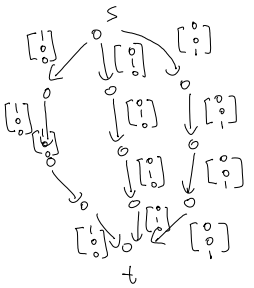
It remains to show that $\det(C) \not\equiv 0$, using our assumption that there are k edge-disjoint paths.

Claim If the source s has k edge-disjoint paths to a receiver t, then $\det(C) \neq 0$ where C is the receiver matrix of the global encoding vectors.

Proof Just use the k edge-disjoint paths from s to t to "forward" the k messages from s to t.

The receiver matrix will be an identity matrix.

So, there is one choice of local encoding coefficients s.t. $\det(C) \neq 0$, which implies that $\det(C)$ is a non-zero polynomial. □



(traditional solution without coding)

To summarize, if there are k edge-disjoint paths between the source and each receiver, then the determinant of each receiver matrix is a non-zero polynomial of the local encoding coefficients.

The determinant polynomials are of low degree, and so Schwartz-Zippel implies that random substitutions will work to ensure that all receiver matrices are of full rank, so that we can decode.

Efficient Encoding

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

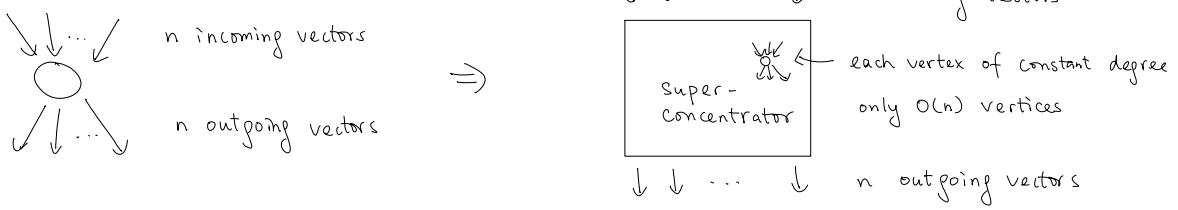
Suppose each vector is of d-dimensional. There are n incoming vector and n outgoing vector.

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.

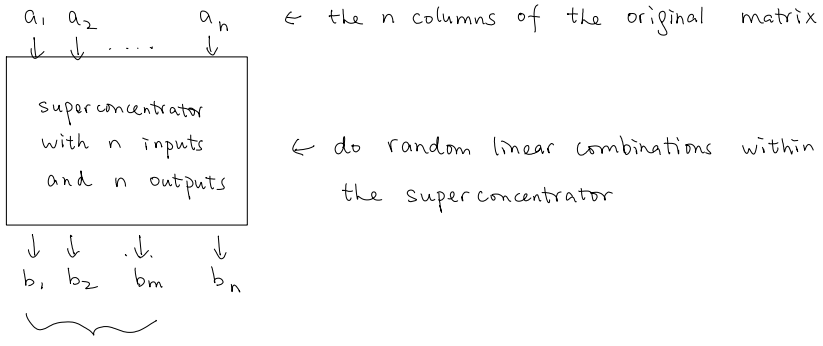
In the homework, you will be asked to use this idea to design a fast algorithm for computing edge connectivities.

Matrix Rank

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.

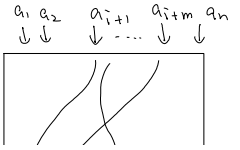
Interestingly, if we just want to compute $\text{rank}(A)$ (i.e only the value), then we can use a superconcentrator to do the job.



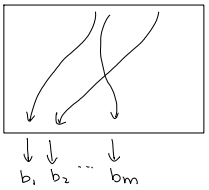
Let $A = \begin{pmatrix} | & | & & | \\ a_1 & a_2 & \cdots & a_n \\ | & | & & | \end{pmatrix}$ and $B = \begin{pmatrix} | & | & & | \\ b_1 & b_2 & \cdots & b_m \\ | & | & & | \end{pmatrix}$. We claim that $\text{rank}(A) = \text{rank}(B)$ w.h.p.

The proof is the same as in network coding.

Say $a_{i+1}, \ldots, a_{i+m}$ are a maximum set of linearly



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 $B = \begin{pmatrix} 1 & 1 & & 1 \\ a_{i+1} & a_{i+2} & \cdots & a_{i+m} \\ 1 & 1 & & 1 \end{pmatrix}$ up to permuting columns, 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.

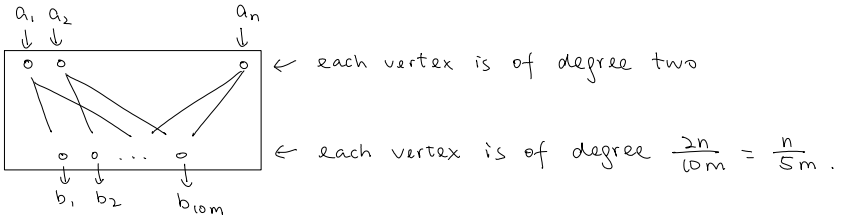
(If the field size is not large enough, we can use an extension field to enlarge the field.)

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

Finding independent columns

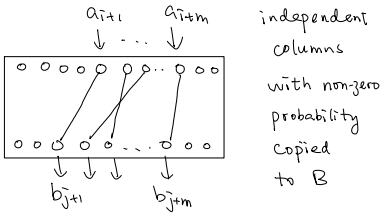
We show an even faster algorithm to compute rank(A), and use it to also find independent columns.

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



By a simple probabilistic argument (will see later), 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.

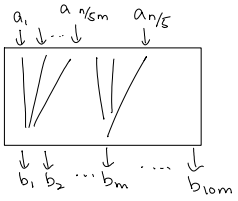
What is the running time? Each column in A is involved in two columns in B.

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.

<u>Application</u>: 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.