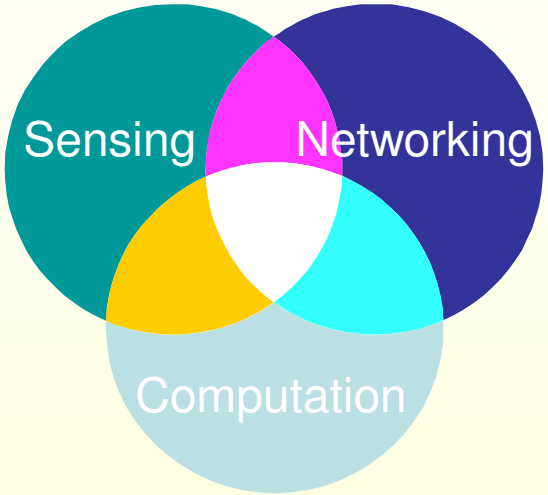
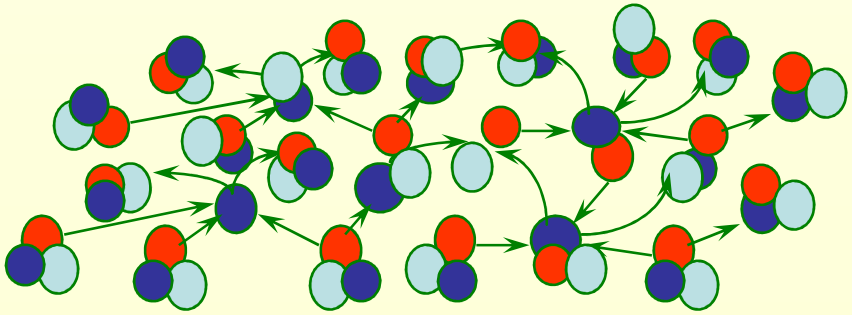
CS321: Gossiping, Coding



Leonidas Guibas
Computer Science Dept.
Stanford University



Gossiping: distributed data fusion

Papers

- [Xiao04] Lin Xiao, Stephen Boyd, Fast Linear Iterations for Distributed Averaging, Systems and Control Letters, 2004.
- [Xiao05] Lin Xiao, Stephen Boyd and Sanjay Lall, A Scheme for Robust Distributed Sensor Fusion Based on Average Consensus, IPSN'05, 2005.
- [Boyd05] S. Boyd, A. Ghosh, B. Prabhakar, D. Shah, Gossip Algorithms: Design, Analysis and Applications, INFOCOM'05.

- Acknowledgement: many slides/figures borrowed from Lin Xiao.

How to Diffuse Information?

- One node has a piece of information that it wants to send to everyone
  - Flood, multi-cast
- Every node has a piece of information that it wants to send to everyone
  - Multi-round flooding
- How do we diffuse information in real life?
  - Gossip!

Uniform Gossip

- We have n nodes in a connected network
- Each node x randomly picks another node y and send to y all the information x has
- After $O(\log n)$ rounds, every node has all the information with high probability
- Totally distributed
- Isotropic protocol

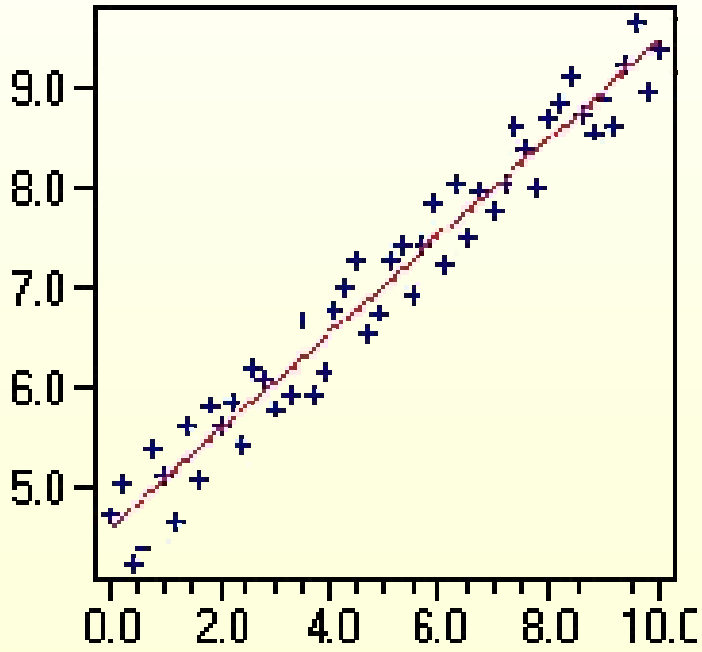
Other Applications

- Load balancing:
  - *N* machines with different work load
  - Goal: balance the load
- Diffusion-based load balancing
  - each machine *x* picks randomly another machine *y* and shift part of its extra load, if any, to *y*
- Good for the case when the work-load of a job is unknown until the job starts.

Using gossiping
for various sensor net computations:
from local to global

Parameter Estimation

- We want to fit a linear model to the sensor data
- E.g., line fitting



for outlier detection

Maximum Likelihood Estimation

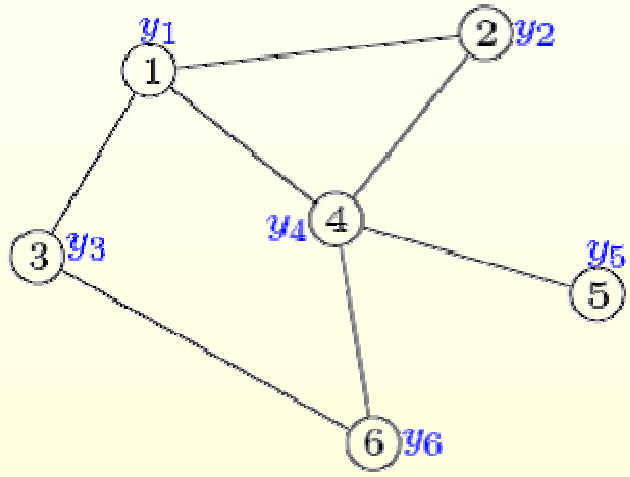
- estimate a vector of unknown parameters $\theta \in \mathbf{R}^m$ with n sensors

$$y_i = A_i\theta + v_i, \qquad i = 1, \ldots, n$$

measurements $y_i \in \mathbf{R}^{m_i}$, noises $v_i \sim \mathcal{N}(0, \Sigma_i)$ independent

- aggregate measurement $(\sum m_i \geq m)$

$$y = A\theta + v = \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_n \end{bmatrix} \theta + \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$



- maximum likelihood estimate given by weighted least-squares solution

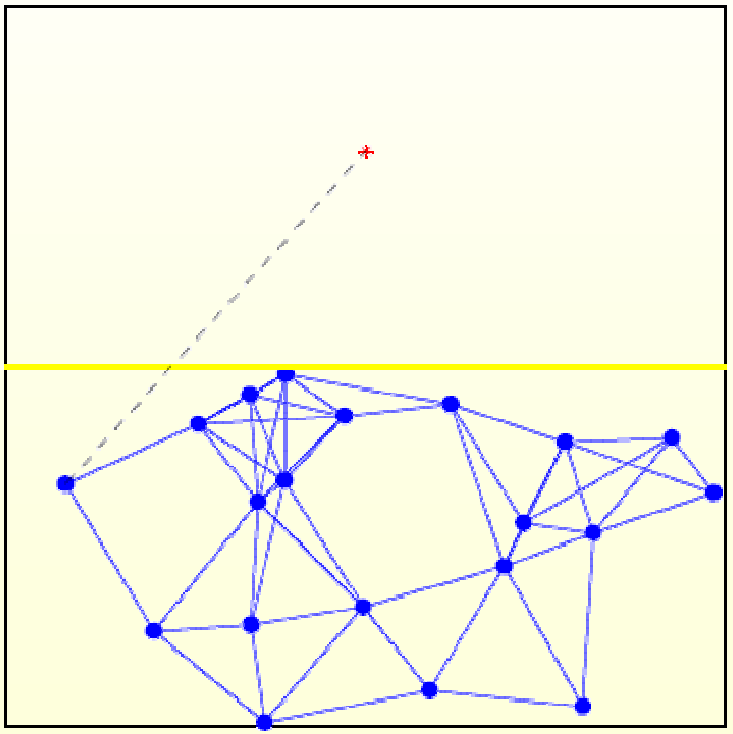
$$\hat{\theta}_{\mathrm{ML}} = \left(A^T\Sigma^{-1}A\right)^{-1} A^T\Sigma^{-1}y, \qquad \text{where } \Sigma = \mathbf{Diag}(\Sigma_1, \ldots, \Sigma_n)$$

Example: Target Localization

- estimate target position (θ_1, θ_2) (red point) within unit square
- 20 range sensors located at (s_{i1}, s_{i2}), $i = 1, \ldots, 20$ (blue spots)
- each sensor measures distance to target r_i, with additive noise $v_i \sim \mathcal{N}(0, 0.1)$
- sensor output

$$y_i = r_i + a_i^T \begin{bmatrix} s_{i1} \\ s_{i2} \end{bmatrix} \approx a_i^T \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} + v_i$$

a_i: unit vector from sensor to target



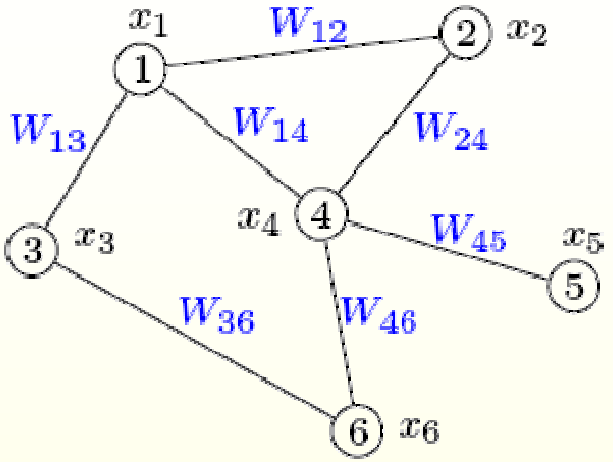
How to Estimate θ?

- Gather all the information and run the centralized maximum likelihood estimate
- Or,
- Use a distributed fusion algorithm:
  - Each sensor exchanges data with its neighbors and carries out local computation, e.g., a least-square estimate
  - Eventually each sensor obtains a good estimation
- Advantages:
  - Completely distributed
  - Robust to link dynamics, only requires a mild assumption on the network connectivity
  - No assumption on routing protocol or any global info

Distributed Average Consensus

- We start with a simple task:
- Goal: compute and deliver to every node the average of the sensor readings by a distributed iterative algorithm
- Assume sensors are synchronized. $x(t)$ is the value of sensor x at time t

Synchronous Algorithm



- compute average $\bar{x} = \frac{1}{n}\sum_i x_i$ (using local communication, iteration)
- each node takes a weighted average of its own and neighbors' values:

$$x_i(t+1) = W_{ii}x_i(t) + \sum_{j \in \mathcal{N}_i} W_{ij}x_j(t)$$

usually $W_{ij} > 0$; but (surprisingly) this is not necessary for all edges

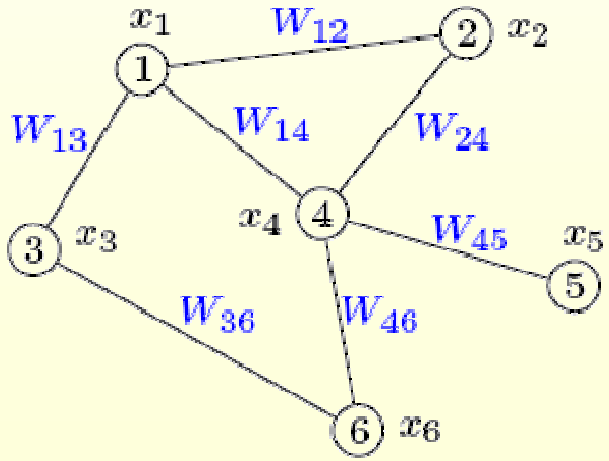
Analysis

- Write the algorithm in a matrix form
- $\boldsymbol{W}$: the weighted adjacency matrix. The value at entry (i,j) is $W_{i,j}$. It is a matrix of size n by n
- $\boldsymbol{x}(\boldsymbol{t})$: the sensor values at time t, a vector of size n
- We know: $\boldsymbol{x}(\boldsymbol{t}+\mathbf{1})=\boldsymbol{W}\boldsymbol{x}(\boldsymbol{t})$
- Inductively, $\boldsymbol{x}(\boldsymbol{t})=\boldsymbol{W}^{\boldsymbol{t}}\boldsymbol{x}(\mathbf{0})$

- We hope the iterative algorithm converges to the correct average

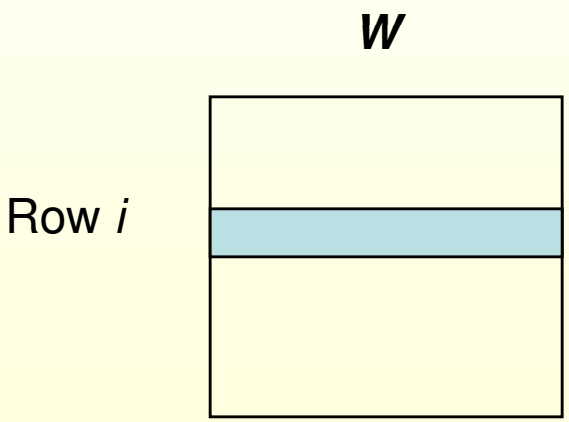
Performance

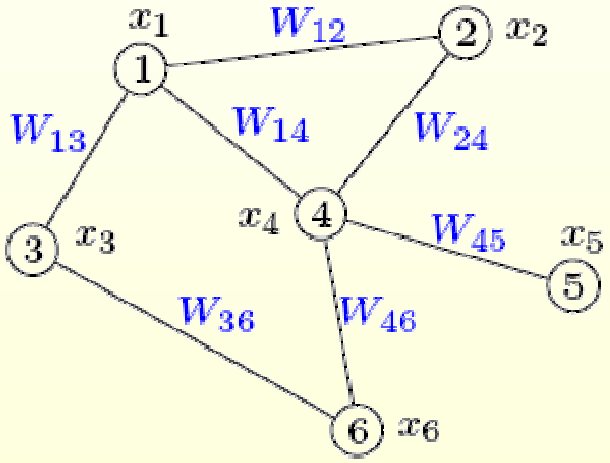
- Questions:
  - Does this algorithm converge?
  - How fast does it converge?
  - How do we choose the weights so that the algorithm converges quickly?



Convergence Condition: Intuition

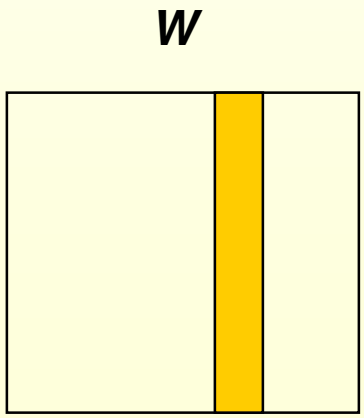
- The vector (1, 1, …, 1) should be a fixed point
- ➔ each row sums up to 1

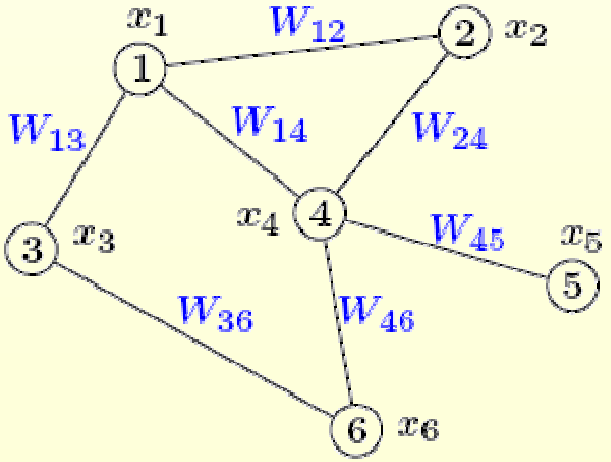




Convergence Condition: Intuition

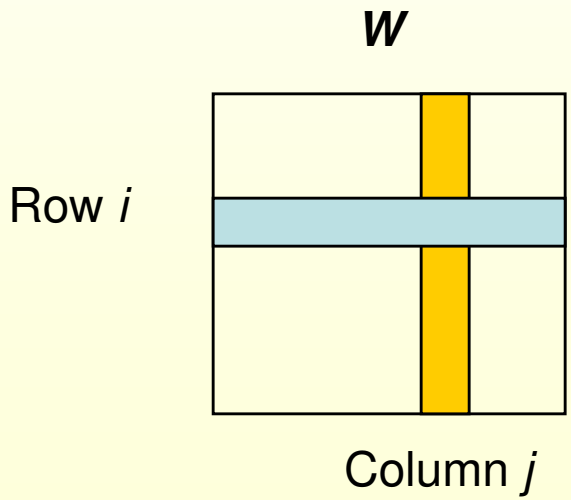
- Think the node values as money. The total money in the system should be conserved
- Mass conservation
- ➔ each column sums up to 1

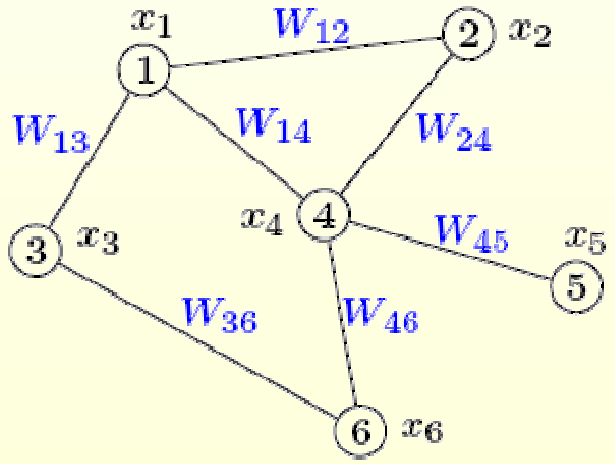




Doubly Stochastic Matrix

- ***W*** must be a doubly stochastic matrix: all the row sum up to 1; and all the columns sum up to 1.





Convergence Condition: Intuition

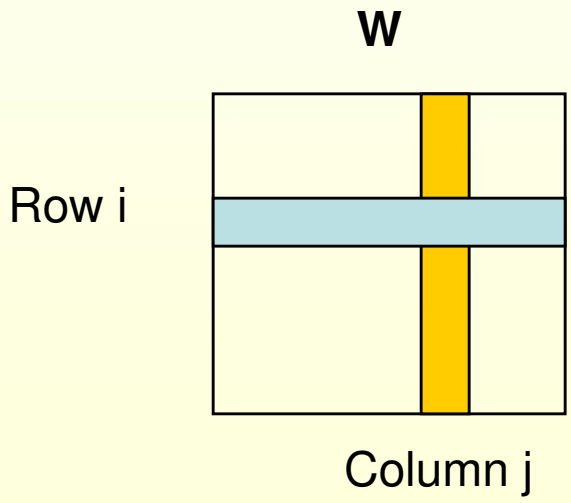
- The algorithm should converge to the average
- Write the average in a matrix form
- Average vector: $1/n\ \mathbf{1}\mathbf{1}^T \boldsymbol{x}(0)$

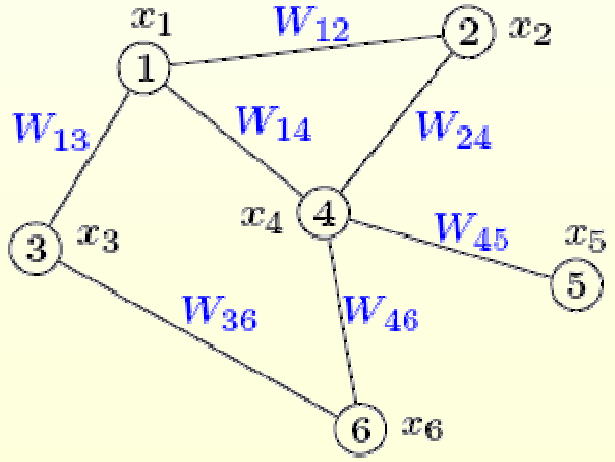
$$\begin{bmatrix} 1/n & 1/n & \dots & 1/n \\ 1/n & 1/n & \dots & 1/n \\ \dots & \dots & \dots & \dots \\ 1/n & 1/n & \dots & 1/n \end{bmatrix}$$

- We want $\boldsymbol{W}^t \to 1/n\ \mathbf{1}\mathbf{1}^T \boldsymbol{x}(0)$, as $t \to \infty$

Convergence Condition

- Theorem: $\lim_{t\to\infty} x(t) = \frac{1}{n}\sum_{i=1}^{n} x_i(0)$ if and only if **W** is a doubly stochastic matrix and the spectral radius of (**W** - **11**^{T}/n) is less than 1.





A detour on a little matrix theory

Matrices, eigenvalues, eigenvectors

- An n by n matrix $\boldsymbol{A}$
- Eigenvalues: $\lambda_1, \lambda_2, \ldots, \lambda_n$. (real or complex numbers)
- Corresponding eigenvectors: $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_n$. (non-zero vectors of size n)
- $\boldsymbol{A}v_i = \lambda_i \boldsymbol{v}_i$
- $\boldsymbol{A}^2 v_i = \boldsymbol{A}(\boldsymbol{A}v_i) = \boldsymbol{A}(\lambda_i \boldsymbol{v}_i) = \lambda_i (\boldsymbol{A}v_i) = \lambda_i^2 \boldsymbol{v}_i$
- Inductively, $\boldsymbol{A}^k \boldsymbol{v}_i = \lambda_i^k \boldsymbol{v}_i$

Spectral Radius

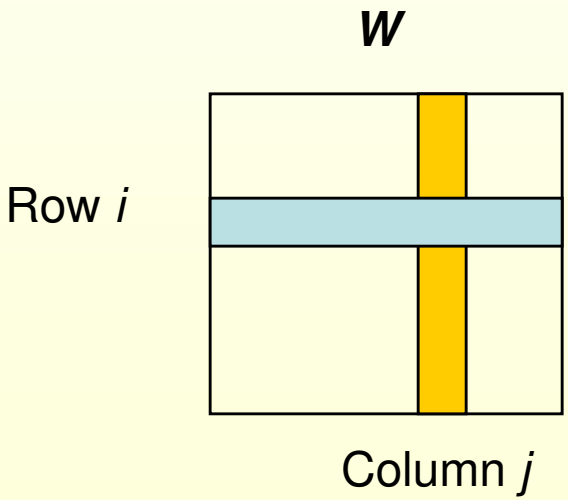
- Spectral radius of M: $\rho(\boldsymbol{A}) = \max|\lambda_i|$.
- Theorem: $\lim_{k\to\infty} A^k = 0$ if and only if $\rho(\boldsymbol{A})<1$

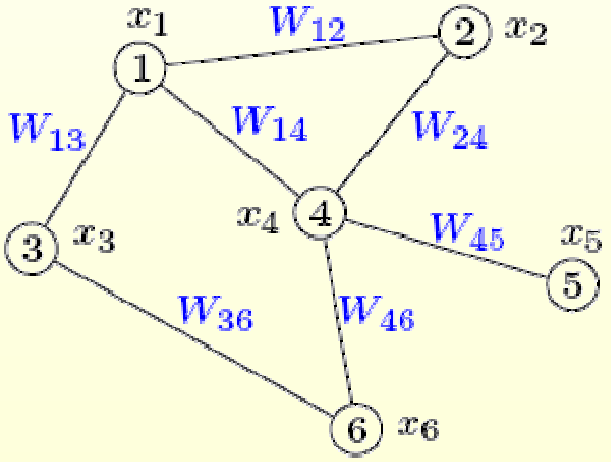
- Proof: (➔) Suppose $\lambda=\rho(\boldsymbol{A})$ with eigenvector $\boldsymbol{v}$
- $0=(\lim \boldsymbol{A}^k)\boldsymbol{v} = \lim \boldsymbol{A}^k\boldsymbol{v} = \lim \lambda^k\boldsymbol{v} = (\lim \lambda^k)\,\boldsymbol{v}$
- Since $\boldsymbol{v}$ is non-zero, $\lim \lambda^k = 0$. This shows $\rho(\boldsymbol{A})<1$
- (🡐) This direction uses Jordan Normal Form

Back to gossiping

Convergence Condition

- Theorem: $\lim_{t\to\infty} x(t) = \frac{1}{n}\sum_{i=1}^{n} x_i(0)$ if and only if ***W*** is a doubly stochastic matrix and the spectral radius of (***W*** - **11**^{T}/*n*) is less than 1





Proof of the Convergence Condition

- Sufficiency: if $\boldsymbol{W}$ is a doubly stochastic matrix and $\rho(\boldsymbol{W} - \mathbf{1}\mathbf{1}^T/n) < 1$, then

$$\lim_{t\to\infty} x(t) = \frac{1}{n}\sum_{i=1}^{n} x_i(0)$$

- Proof:
  1. W is doubly stochastic. Thus

$$1^T W = 1^T, \qquad W1 = 1,$$

  2. Now we have

$$\begin{aligned} W^t - 11^T/n &= W^t\left(I - \mathbf{1}\mathbf{1}^T/n\right) \\ &= W^t\left(I - \mathbf{1}\mathbf{1}^T/n\right)^t \\ &= \left(W\left(I - 11^T/n\right)\right)^t \\ &= \left(W - \mathbf{1}\mathbf{1}^T/n\right)^t, \end{aligned}$$

  3. Since $\rho(\boldsymbol{W} - \mathbf{1}\mathbf{1}^T/n) < 1$, $\lim_{t\to\infty} W^t = \frac{11^T}{n}$.

Convergence Rate

- **convergence factor**

$$\rho(W - \mathbf{1}\mathbf{1}^T/n) = \max_{i=2,\ldots,n} |\lambda_i(W)|$$

asymptotically

$$\|x(t) - \bar{x}\|_2 \leq \rho^t \, \|x(0) - \bar{x}\|_2$$

The smaller the better.

- associated **mixing time**

$$\tau = \frac{1}{\log(1/\rho)}$$

(asymptotic) number of steps for the error to decrease by factor e

Fastest Iterative Algorithm?

- Given a graph, find the weight function ***W*** such that the iterative algorithm converges fastest

$$\begin{array}{ll} \text{minimize} & \rho(W - 11^T/n) \\ \text{subject to} & W \in \mathcal{S}, \quad 1^T W = \mathbf{1}^T, \quad W1 = 1, \end{array}$$

- Theorem (Xiao & Boyd 04): When the matrix ***W*** is symmetric, the above optimization problem can be formulated by a semi-definite programming and can be solved efficiently

negative weights are possible!

Choosing the Weights

- constant weight on all edges, *e.g.*, **max-degree weights**

$$W_{ij} = \frac{1}{d_{\max}}, \qquad \{i,j\} \in \mathcal{E}$$

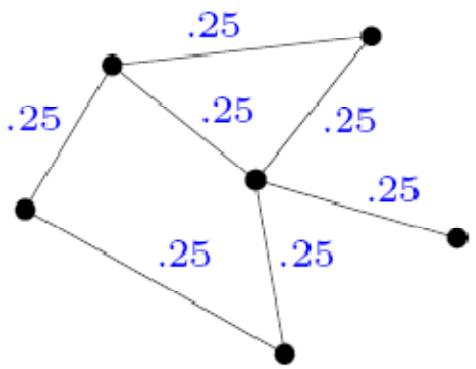
(self-weights given by $W_{ii} = 1 - \sum_{j \in \mathcal{N}_i} W_{ij}$)

- **Metropolis weights** (only needs local information)

$$W_{ij}(t) = \begin{cases} \dfrac{1}{\max\{d_i(t), d_j(t)\}} & \text{if } \{i,j\} \in \mathcal{E}(t), \\ 1 - \displaystyle\sum_{\{i,k\} \in \mathcal{E}(t)} W_{ik}(t) & \text{if } i = j, \\ 0 & \text{otherwise.} \end{cases}$$

Example: Weight Selection

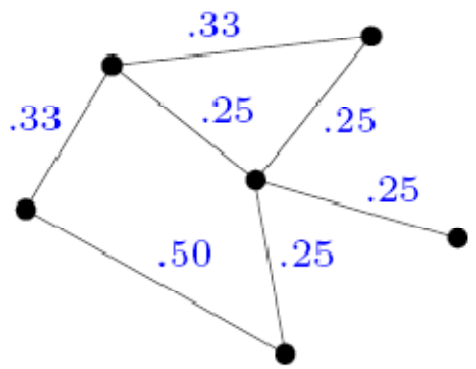
max-degree weights



$\rho^{\mathrm{md}} = 0.78$

$\tau^{\mathrm{md}} = 4.02$

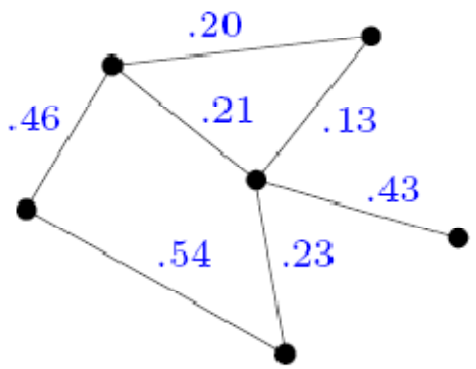
Metropolis weights



$\rho^{\mathrm{mh}} = 0.77$

$\tau^{\mathrm{mh}} = 3.91$

fastest weights



$\rho^{\star} = 0.72$

$\tau^{\star} = 3.06$

Extension to changing topologies

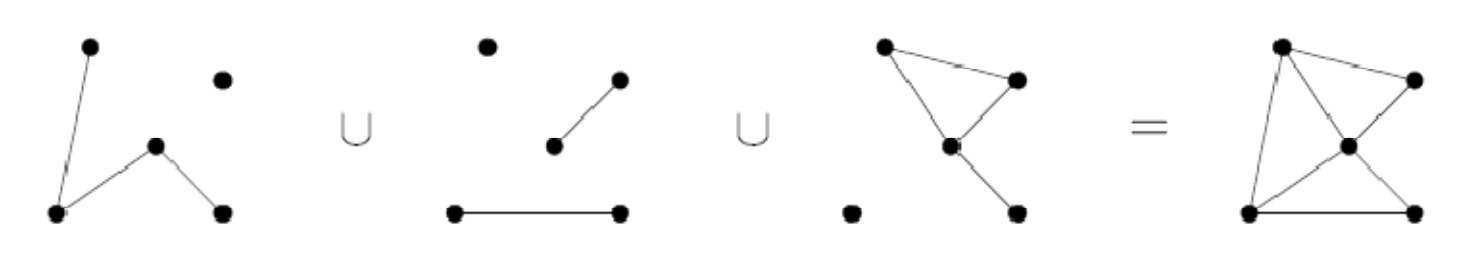
Changing Topologies

- The sensor network topology changes over time
  - Link failure
  - **Mobility**
  - Power constraints
  - Channel fading

- However, the distributed fusion algorithm only assumes a mild condition on network connectivity --- the network is "**connected in the long run**"

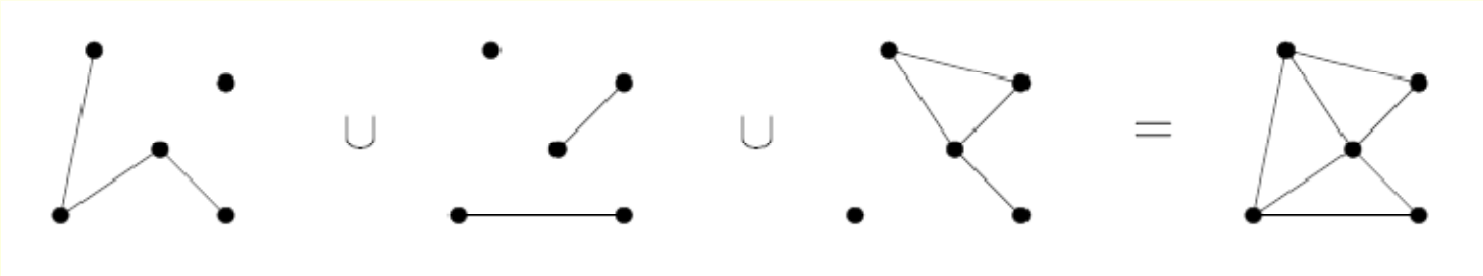
Changing Topologies

- The communication graph $G(t)$ is time-varying.
- For n nodes, there are only finitely many communication graphs, and finitely many weight functions
- There are a subset of graphs that appear infinitely many times
- If the collection of graphs that appear infinitely many times are **jointly connected**, then the algorithm converges (w. Metropolis or max-d weights)



Changing Topologies

- This a very mild condition on connectivity
- Many links can fail permanently
- We only require that a connected graph “survives” in the sequence of (possibly disconnected) graphs



Choice of Weights

- Maximum-degree weights

$$W_{ij}(t) = \begin{cases} \dfrac{1}{n} & \text{if } \{i,j\} \in \mathcal{E}(t) \\ 1 - \dfrac{d_i(t)}{n} & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

- Metropolis weights (only need real-time local information)

$$W_{ij}(t) = \begin{cases} \dfrac{1}{1+\max\{d_i(t), d_j(t)\}} & \text{if } \{i,j\} \in \mathcal{E}(t) \\ 1 - \displaystyle\sum_{\{i,k\} \in \mathcal{E}(t)} W_{ik}(t) & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

Extension to parameter estimation

Maximum Likelihood Estimation

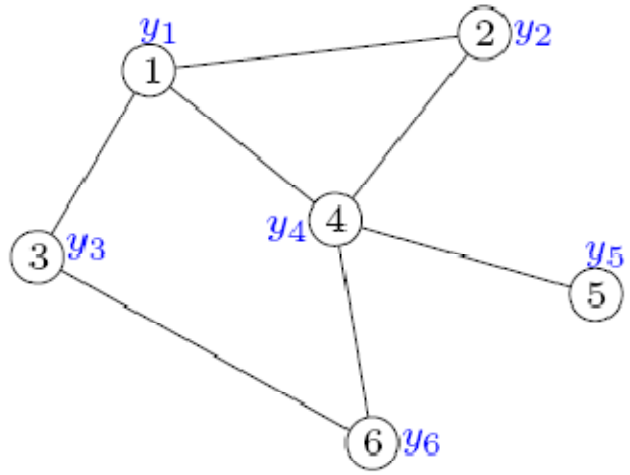
- estimate a vector of unknown parameters $\theta \in \mathbf{R}^m$ with n sensors

$$y_i = A_i\theta + v_i, \qquad i = 1, \ldots, n$$

measurements $y_i \in \mathbf{R}^{m_i}$, noises $v_i \sim \mathcal{N}(0, \Sigma_i)$ independent

- aggregate measurement $(\sum m_i \geq m)$

$$y = A\theta + v = \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_n \end{bmatrix} \theta + \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$



- maximum likelihood estimate given by weighted least-squares solution

$$\hat{\theta}_{\mathrm{ML}} = \left(A^T\Sigma^{-1}A\right)^{-1} A^T\Sigma^{-1}y, \qquad \text{where } \Sigma = \mathbf{Diag}(\Sigma_1, \ldots, \Sigma_n)$$

Distributed Parameter Estimation

- A sensor node i knows

$$y_i = A_i\theta + v_i, \qquad i = 1, \ldots, n$$

- Goal: we want to evaluate in a distributed fashion

$$\hat{\theta}_{\mathrm{ML}} = \left(A^T\Sigma^{-1}A\right)^{-1} A^T\Sigma^{-1}y, \qquad \text{where } \Sigma = \mathbf{Diag}(\Sigma_1, \ldots, \Sigma_n)$$

- Idea: use the average consensus algorithm.

Distributed Parameter Estimation

- distributed computation of $\hat{\theta}_{\mathrm{ML}} = \left(A^T \Sigma^{-1} A\right)^{-1} A^T \Sigma^{-1} y$

define $$P = [A_1^T \dots A_n^T] \begin{bmatrix} \Sigma_1 & & \\ & \ddots & \\ & & \Sigma_n \end{bmatrix}^{-1} \begin{bmatrix} A_1 \\ \vdots \\ A_n \end{bmatrix} = \sum_{i=1}^{n} A_i^T \Sigma_i^{-1} A_i$$

$$q = [A_1^T \dots A_n^T] \begin{bmatrix} \Sigma_1 & & \\ & \ddots & \\ & & \Sigma_n \end{bmatrix}^{-1} \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} = \sum_{i=1}^{n} A_i^T \Sigma_i^{-1} y_i$$

therefore

$$\hat{\theta}_{\mathrm{ML}} = P^{-1} q = \left(\sum_{i=1}^{n} A_i^T \Sigma_i^{-1} A_i \right)^{-1} \sum_{i=1}^{n} A_i^T \Sigma_i^{-1} y_i$$

- **idea:** distributed average consensus of matrices and vectors (entry-wise)

$P_i(0) = A_i^T \Sigma_i^{-1} A_i$ ($m \times m$ composite information matrix)

$q_i(0) = A_i^T \Sigma_i^{-1} y_i$ ($m \times 1$ composite measurement vector)

- distributed average consensus (same form as in scalar case)

$$\begin{aligned} P_i(t+1) &= W_{ii}(t)P_i(t) + \sum_{j\in\mathcal{N}_i(t)} W_{ij}(t)P_j(t) \\ q_i(t+1) &= W_{ii}(t)q_i(t) + \sum_{j\in\mathcal{N}_i(t)} W_{ij}(t)q_j(t) \end{aligned}$$

- use distributed average consensus to compute

$$\lim_{t\to\infty} P_i(t) = \frac{1}{n}\sum_{i=1}^{n} A_i^T\Sigma_i^{-1}A_i = \frac{1}{n}P$$

$$\lim_{t\to\infty} q_i(t) = \frac{1}{n}\sum_{i=1}^{n} A_i^T\Sigma_i^{-1}y_i = \frac{1}{n}q$$

- each node can **eventually** compute

$$\hat{\theta}_{\mathrm{ML}} = P_i(\infty)^{-1}q_i(\infty) = P^{-1}q$$

Intermediate Estimates

- instead of waiting for convergence, use

$$\hat{\theta}_i(t) = P_i(t)^{-1} q_i(t)$$

available at node i as soon as $P_i(t)$ invertible

Properties

- universal data structure for storage and communication

$$P_i(t) \in \mathbf{R}^{m \times m}, \; q_i(t) \in \mathbf{R}^m$$

  independent of local dimension m_i (of A_i and y_i)

- isotropic protocol (diffusion)
  - taking weighted average of neighbors' data (Metropolis weights)
  - no explicit routing/broadcasting involved
  - convergence under dynamically changing topology

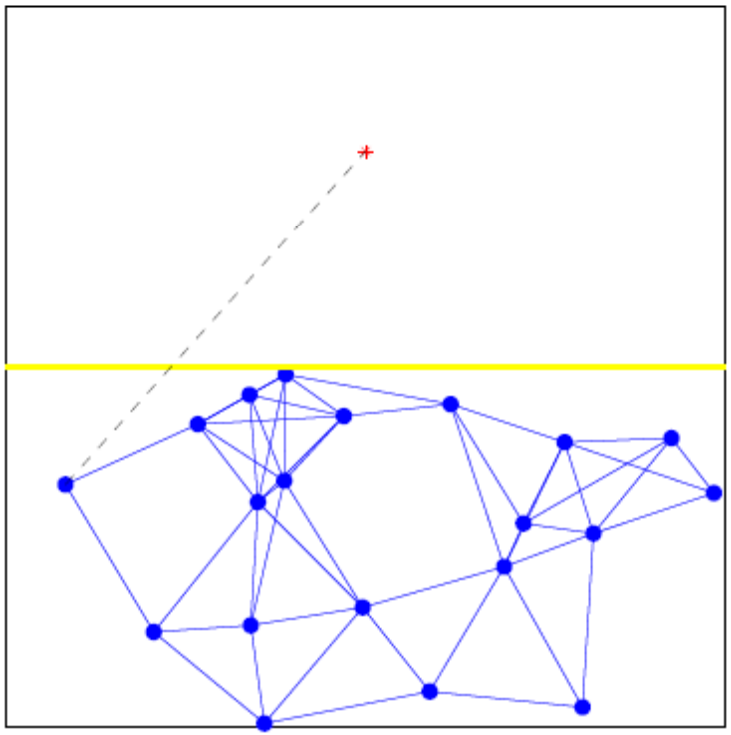
- intermediate estimates $\hat{\theta}_i(t) = P_i(t)^{-1} q_i(t)$
  - available at each node whenever $P_i(t)$ invertible
  - always unbiased
  - error covariances converge to global optimal

Localization

- estimate target position (θ_1, θ_2) (red point) within unit square
- 20 range sensors located at (s_{i1}, s_{i2}), $i = 1, \ldots, 20$ (blue spots)
- each sensor measures distance to target r_i, with additive noise $v_i \sim \mathcal{N}(0, 0.1)$
- sensor output

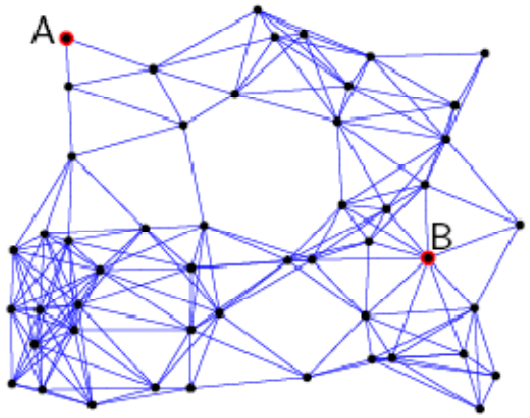
$$y_i = r_i + a_i^T \begin{bmatrix} s_{i1} \\ s_{i2} \end{bmatrix} \approx a_i^T \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} + v_i$$

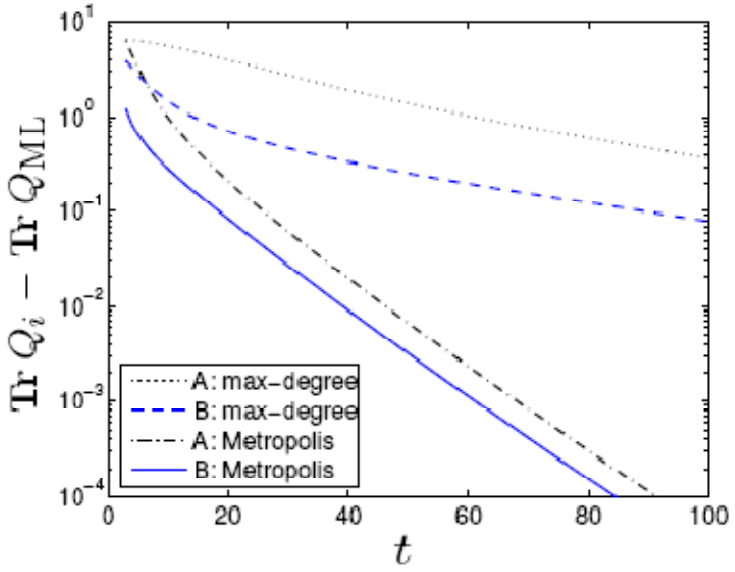
a_i: unit vector from sensor to target

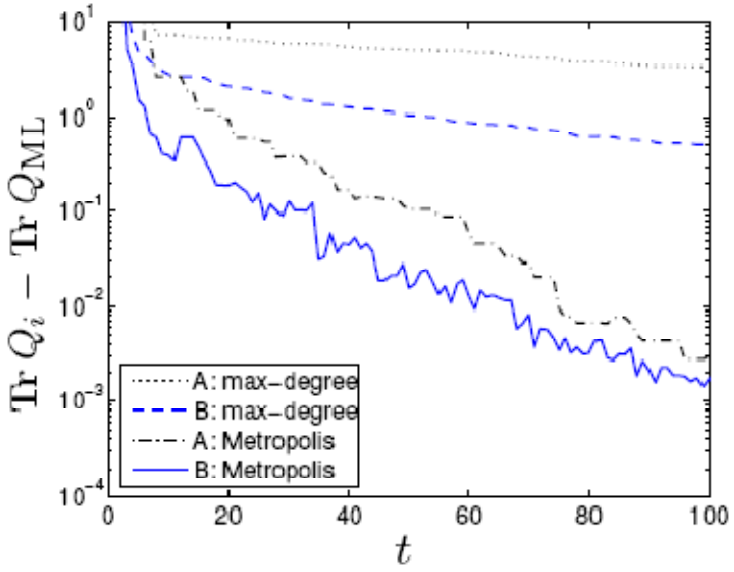


An Example

- 50 sensors, 200 communication links
- $\theta \in \mathbf{R}^5$, measurements $y_i = a_i^T \theta + v_i$
  a_i uniform distribution on unit sphere
  $v_i \sim \mathcal{N}(0, 1)$
- convergence of mean-squares error
  - left: distributed fusion on fixed graph
  - right: links fail (iid) with prob. $3/4$







Random gossip model

Random Gossip

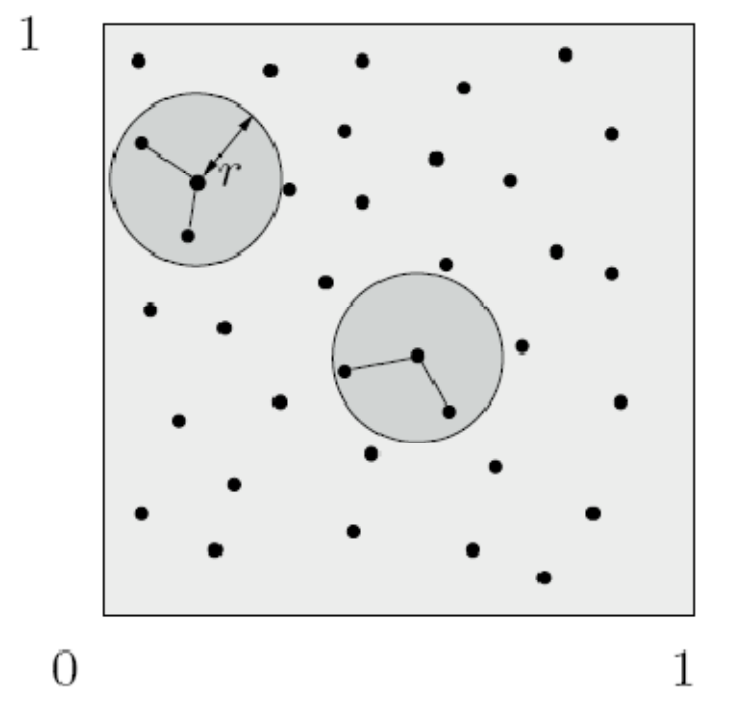
- Completely asynchronous. No synchronized clock is needed
- At each time, a node can only talk to **one** other node
- Distributed average consensus: each node chooses one node with some **probability distribution** and compute the average
- **Natural averaging algorithm**: each node uniformly randomly picks a neighbor and compute the avg.
- Again, one can find the **optimal averaging** distribution by convex programming s.t. the algorithm converges fastest

Random geometric graphs

- $G^d(n, r)$: place n nodes uniformly random in a d-dimensional cube and connect two nodes if they are within distance r.

- Bad news: the natural averaging algorithm converges about the same order as the optimal one ➔ both are slow.

- Good news: no need to optimize. The natural averaging is a local and distributed algorithm with optimal performance.



Internet

- Preferential attachment model: a new comer connects by an edge to the existing nodes with probability proportion to the degree
- “Rich get richer”
- The graph obtained is an expander:
  - spectral gap is a constant;
  - the second largest eigenvalue is small enough;
  - random walk mixes fast;
- Optimal averaging algorithm has an averaging time $O(\log 1/\varepsilon)$, independent of the graph size
- Averaging on P2P network is extremely fast

Gossip Summary

- One of the few examples that are so robust to topological changes
- Many applications on similar problems
- Distributed optimization

Coding and its applications to sensor networks

Paper

- **[Dimakis05]** A. G. Dimakis, V. Prabhakaran and K. Ramchandran, **Ubiquitous Access to Distributed Data in Large-Scale Sensor Networks through Decentralized Erasure Codes**, Symposium on Information Processing in Sensor Networks (IPSN'05), April, 2005.

Why Coding?

- Information compression
- Robustness to errors (error correcting codes)

Source Coding

- Compression
- What is the minimum number of bits to represent certain information? What is a measure of information?

- Entropy, Information theory

Channel Coding

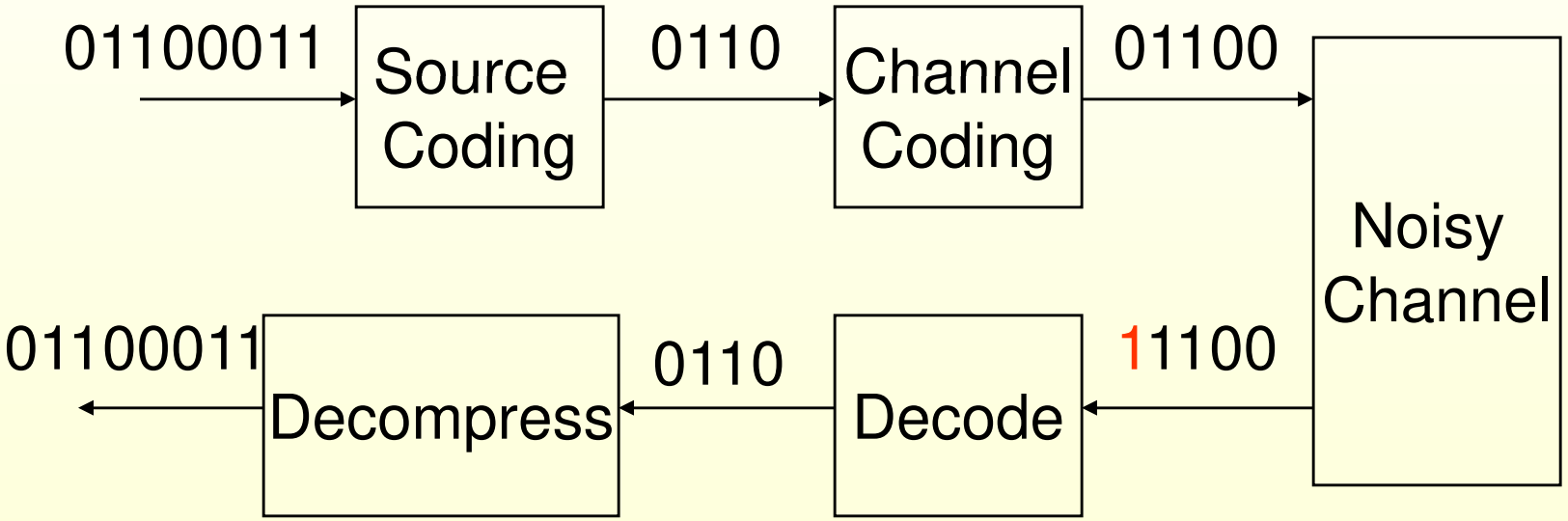
- Achieve fault tolerance.

- Transmit information through a noisy channel.
- Storage on a disk. Certain bits may be flipped.

- Goal: recover the original information.
- How? duplicate information.

Source Coding and Channel Coding

- Source coding and channel coding can be combined



Coding in Sensor Networks

- Compression
  - Sensors generate too much data
  - Nearby sensor readings are correlated
- Fault tolerance
  - Communication failures. Corrupted messages by a noisy channel
  - Node failures – fault tolerance storage
  - Adversaries inject false information

Channels

- The media through which information is passed from a sender to a receiver
- Binary symmetric channel: each symbol is flipped with probability p
- Erasure channel: each symbol is replaced by a "?" with probability p

- We first focus on binary symmetric channel

Encoding and Decoding

- **Encoding:**
- Input: a string of length k, “data”
- Output: a string of length $n>k$, “codeword”

- **Decoding:**
- Input: some string of length n (might be corrupted)
- Output: the original data of length k

Error Detection and Correction

- Error detection: detect whether a string is a valid codeword.
- Error correction: correct it to a valid codeword.

- Maximum likelihood decoding: find the codeword that is “closest” in Hamming distance, i.e., with minimum # flips.

- How to find it?
- For small size code, store a codebook. Do table lookup.
- NP-hard in general.

Scheme 1: Repetition

- Simplest coding scheme one can come up with.
- Input data: 0110010
- Repeat each bit 11 times.
- Now we have
- 00000000000111111111111111111111100000000000000000000001111111111100000000000
- Decoding: do majority vote.
- Detection: when the 10 bits don’t agree with each other.
- Correction: up to 5 bits of error.

Scheme 2: Parity-Check

- Add one bit to do parity check.
- Sum up the number of “1”s in the string. If it is even, then set the parity check bit to 0; otherwise set the parity check bit to 1.
- Eg. 0010111010, 1110111111.

- Sum of 1’s in the codeword is even.
- 1-bit parity check can detect 1-bit error. If one bit is flipped, then the sum of 1s is odd.
- But cannot detect 2 bits error, nor can correct 1-bit error.

More on Parity-Checks

- Encode a piece of data into codeword.
- Not every string is a codeword.
- After 1 bit parity check, only strings with even 1s are valid codeword.
- Thus we can detect error.

- Minimum Hamming distance between any two codewords is 2.
- Suppose we make the min Hamming distance larger, then we can detect more errors and also correct errors.

Scheme 3: Hamming Codes

- Intuition: generalize the parity bit and organize them in a nice way so that we can detect and correct more errors.
- Lower bound: If the minimum Hamming distance between two code words is k, then we can detect at most k-1 bits error and correct at most $\lfloor k/2 \rfloor$ bits error.
- Hamming code (7,4): adds three additional check bits to every four data bits of the message to correct any single-bit error, and detect all two-bit errors.

Hamming Code (7, 4)

- Coding: multiply the data with the encoding matrix.

$$H_e := \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix}$$

- Decoding: multiply the codeword with the decoding matrix.

$$H_d := \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

An Example: Encoding

- Input data:

$$\mathbf{p} = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \end{pmatrix}$$

- Codeword:

$$H_e\mathbf{p} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} = \mathbf{r}$$

Original data is preserved

Systematic code: the first *k* bits is the data.

An Example: Decoding

- Decode:

$$H_d\mathbf{r} = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

- Now suppose there is an error at the *i*-th bit.
- We received $\mathbf{r} + \mathbf{e}_i$
- Now decode:

$$H_d(\mathbf{r} + \mathbf{e}_i) = H_d\mathbf{r} + H_d\mathbf{e}_i$$

$$H_d\mathbf{r} + H_d\mathbf{e}_i = 0 + H_d\mathbf{e}_i = H_d\mathbf{e}_i$$

- This picks up the *i*-th column of the decoding vector!

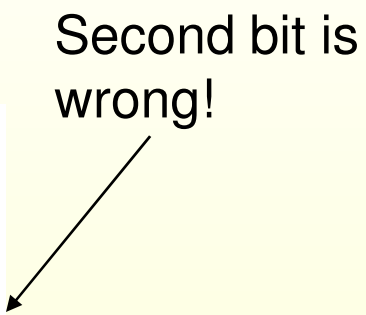
An Example: Decoding

- Suppose

$$\mathbf{s} = \mathbf{r} + \mathbf{e}_2 = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

- Decode:

$$H_d\mathbf{s} = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$



- Data more than 4 bits? Break it into chunks and encode each chunk.

Reed-Solomon Codes

- Most commonly used code, in CDs/DVDs.
- Linear code, handles bursty errors.
- Use a large alphabet and algebra.

- Take an alphabet of size $q > n$ and n distinct elements $\alpha_1, \ldots, \alpha_n \in F_q$
- Input message of length k: $c_0, \ldots, c_{k-1}$
- Define the polynomial $C(x) \stackrel{\text{def}}{=} \sum_{j=0}^{k-1} c_j x^j$

- The codeword is $\langle C(\alpha_1), \ldots, C(\alpha_n) \rangle$

Reed-Solomon Codes

- Rephrase the encoding scheme.
- Unknowns (variables): the message of length *k*

$$c_0, \ldots, c_{k-1}$$

- What we know: some equations on the unknowns.

$$\langle C(\alpha_1), \ldots, C(\alpha_n) \rangle \qquad C(x) \stackrel{\text{def}}{=} \sum_{j=0}^{k-1} c_j x^j$$

- Each of the coded bits gives a linear equation on the *k* unknowns. ➔ a linear system.
- How many equations do we need to solve it?

- We only need length *k* coded values to solve for all the unknowns.

Reed-Solomon Codes

- Write the linear system by matrix form:

$$\begin{bmatrix} 1 & \alpha_1 & \alpha_1^2 & \alpha_1^{k-1} \\ 1 & \alpha_2 & \alpha_2^2 & \alpha_2^{k-1} \\ \dots & \dots & \dots & \dots \\ 1 & \alpha_k & \alpha_k^2 & \alpha_k^{k-1} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \dots \\ c_{k-1} \end{bmatrix} = \begin{bmatrix} C(\alpha_1) \\ C(\alpha_2) \\ \dots \\ C(\alpha_k) \end{bmatrix}$$

- This is the Van der Monde matrix. So it's invertible.

- This code can tolerate *n-k* errors.
- Any *k* bits can recover the original message.

Coding in sensor network storage

Use Coding for Fault Tolerance

- If a sensor dies, we lose the data.
- For fault tolerance, we have to replicate the data so that we can recover the data from other sensors.
- Straight-forward solution: duplicate it at other places.
- Storage size goes up!

- Use coding to keep storage size about the same.
- What we pay: decoding cost.

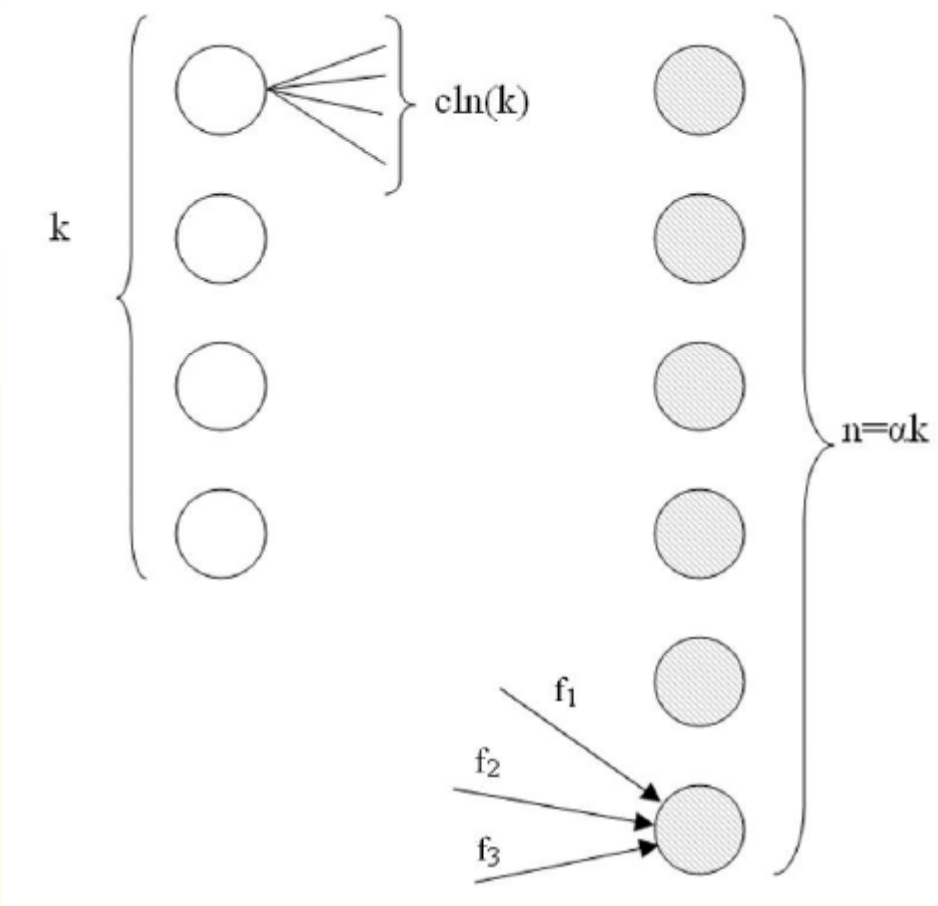
Problem Setup

- Setup: we have k data nodes (nodes w. data), and $n > k$ storage nodes (data nodes may also be storage nodes). Here $n = \Theta(k)$.
- Each data node generates one piece of data.
- Each storage node only stores one piece of (coded) data.
- We want to recover data by using any k storage nodes.

- Sounds familiar? Reed Solomon code.
- But it is centralized -- we need all the k inputs to generate the coded information.

Distributed Random Linear Code

- Each node sends its data to $m = O(\log k)$ random storage nodes.
- A storage node may receive multiple pieces of data $c_1, c_2, \ldots c_k$, but it stores a random combination of them, e.g., $a_1c_1+a_2c_2+\ldots+a_kc_k$, where a's are random coefficients.



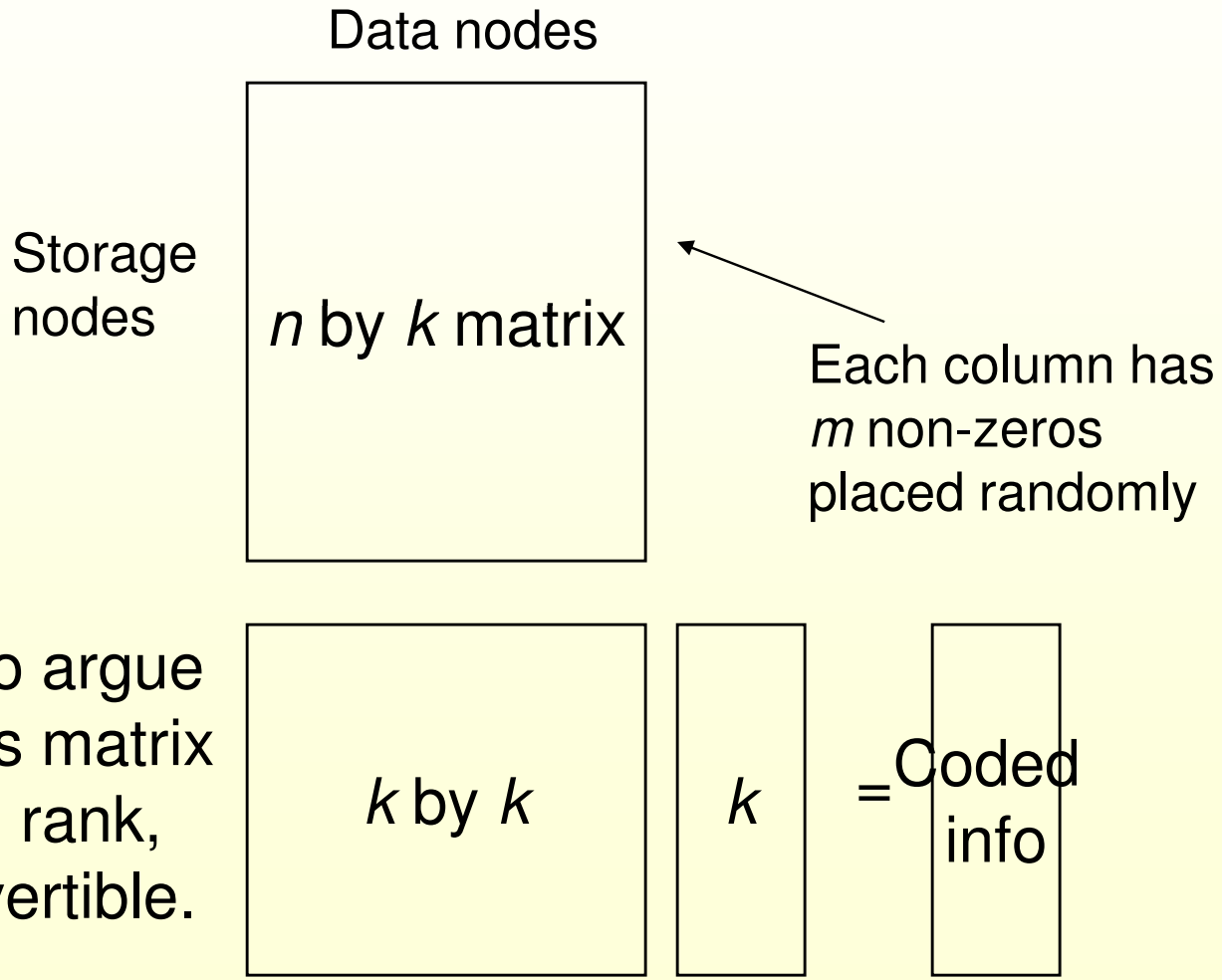
Coding and Decoding

- Storage size keeps almost the same as before.
- The random coefficients can be generated by a pseudo-random generator. Even if we store the coefficients, their size is modest.

- Claim: we can recover the original k pieces of data from any k storage nodes.
- Think of the original data as unknowns (variables).
- Each storage node provides a linear equation on the unknowns $a_1c_1+a_2c_2+\ldots+a_kc_k = s$.
- Now we take k storage nodes and look at the linear system.

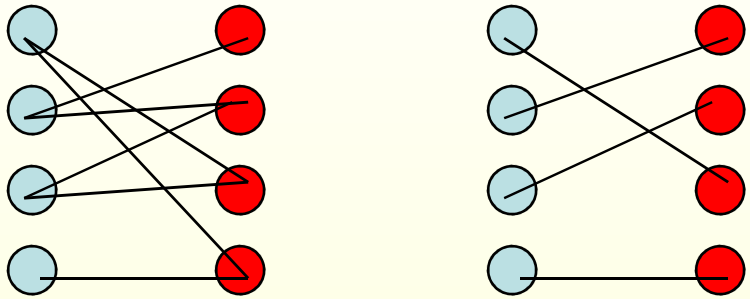
Coding and Decoding

- Take arbitrary *k* storage nodes.



Main Theorem

- A bipartite graph $G=(X, Y)$, $|X|=k$, $|Y|=k$.
- X: the k data nodes; Y: the k storage nodes.



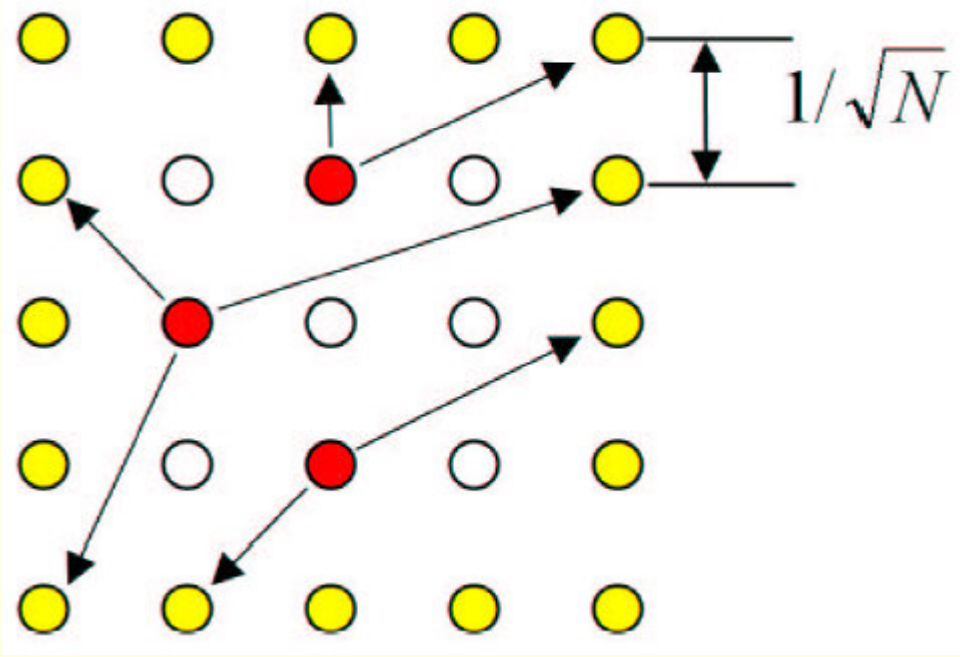
- Edmond's theorem: the matrix has full rank if the bipartite graph has a perfect matching.
- Now, we only need to show that the bipartite graph G has a perfect matching with high probability.

Main Theorem

- Upper bound: if a data node picks $O(\log k)$ storage randomly, the bipartite graph G has a perfect matching with high probability.
- Lower bound: $\Omega(\log k)$ is necessary.
- Proof:
  - Any storage node has to have at least one piece of data.
  - Otherwise, the matrix has a zero row!
  - Throw data randomly to cover all the storage nodes.
  - Coupon collector problem: each time we get a random coupon. In order to collect all n different types of coupon, with high probability one has to get in total $\Omega(n \log n)$ coupons.

Perimeter Storage

- Potential users outside the network have easy access to perimeter nodes; Gateway nodes are positioned on the perimeter.



Pros and Cons

- No extra infrastructure, only a point-to-point routing scheme is needed.
- Robust to errors – just take *k* good copies.
- Fault tolerance – sensors die? Fine…
- No centralized processing, no routing table or global knowledge of any sort.
- Very resilient to packet loss due to the random nature of the scheme.
- Achieves certain data privacy. If the coding scheme (the random coefficients) is kept from the adversary, the adversary only sees random data.

Pros and Cons

- Information is coded, in other words, scrambled.
- Have to decode the whole *k* pieces, even only 1 piece of data is desired.
- Doesn’t explore locality – usually we don’t go to arbitrary *k* storage nodes, we go to the closest *k* nodes.

Summary

- Combing coding idea with sensor storage and communication schemes is a very promising area.

- Distributed coding schemes.
- Locality-aware (geometry-aware) coding schemes.

Network coding

References

- A 6-page network coding introduction:

C. Fragouli, J. Le Boudec, Jorg Widmer, **Network coding: an instant primer**.

- Network coding webpage:

http://tesla.csl.uiuc.edu/~koetter/NWC/

- A book: "Network coding theory" (Raymond Yeung, Ning Cai)

Existing Networks

- Independent data streams sharing the same network resources
  - Packets over the Internet
  - Signals in a phone network
  - An analogy: cars sharing a highway
- Information flows are separated
- What if we mix them?

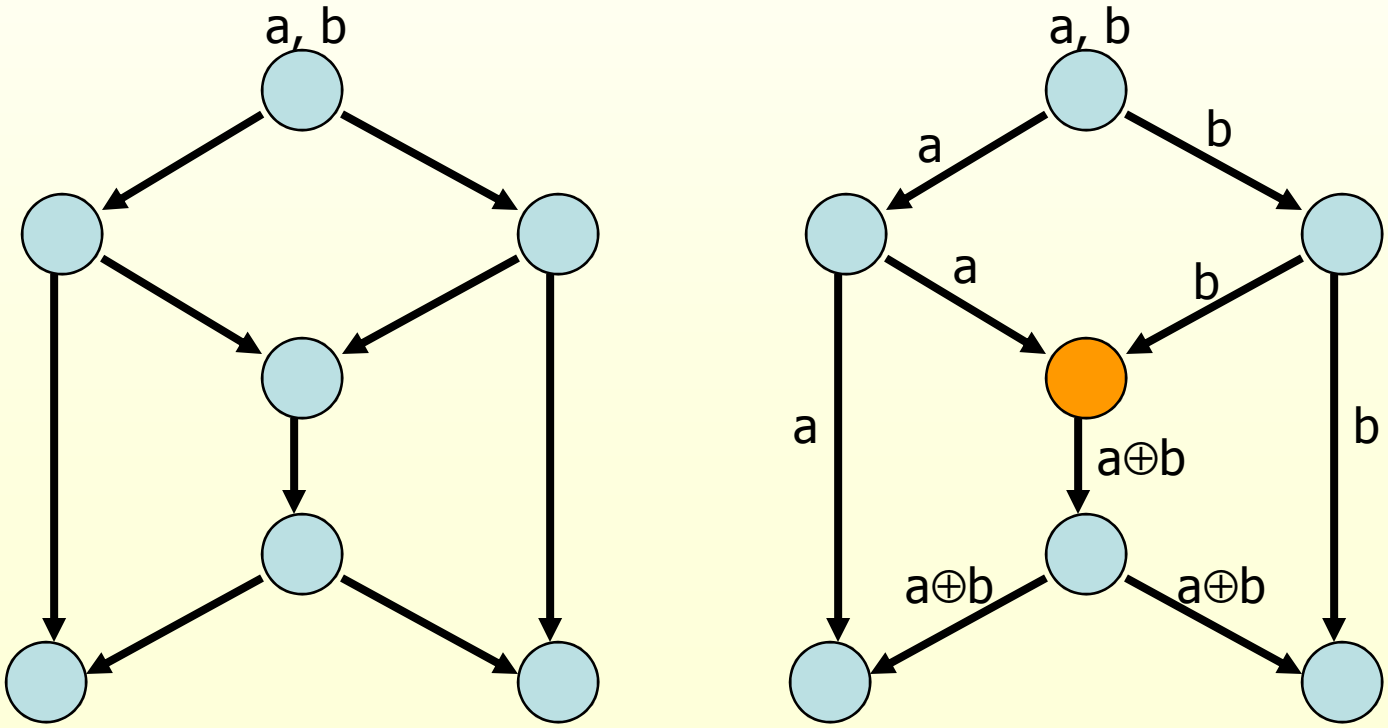
Why Mix Flows?

- The core notion of network coding is to allow and encourage mixing of data at intermediate network nodes.

- R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, "Network Information Flow", *IEEE Transactions on Information Theory*, IT-46, pp. 1204-1216, 2000.

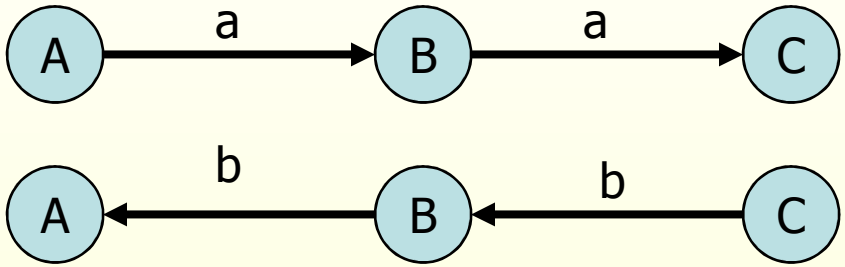
Network Coding Increases Throughput

- Butterfly network
- Multi-cast: throughput increases from 1 to 2.

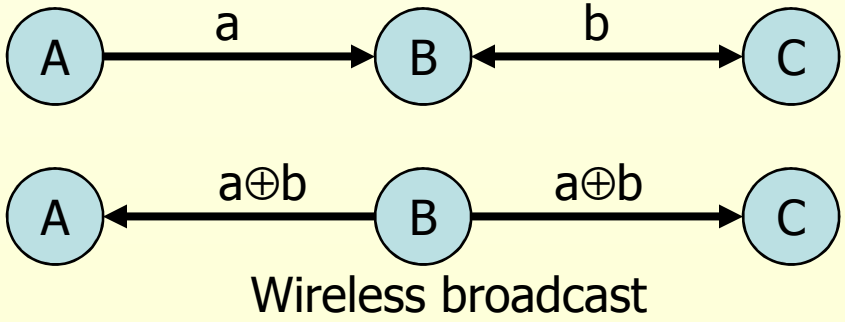


Network Coding Saves Energy & Reduces Delay in Wireless Networks

- A wants to send packet a to C.
- C wants to send packet b to A.



- B performs coding



Linear Coding is Enough

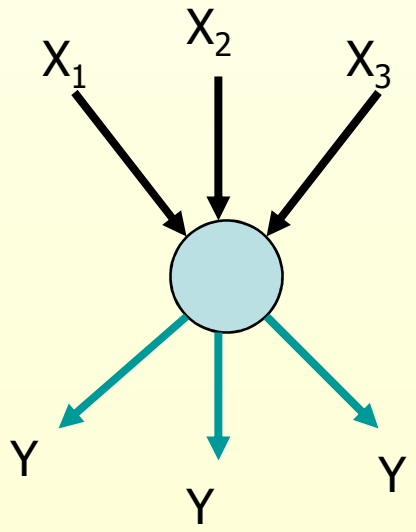
- Linear code: basically take linear combinations of input packets.
  - Not concatenation!
  - 3a+5b: has the same length as a, b.
  - + maybe XOR (+ in Z_2).
- Even better: random linear coding is enough.
  - Choose coding coefficients randomly.

Encode

- Original packets: M_1, M_2, $\cdots$, M_n.
- An incoming packet is a linear combination of the original packets
- $X=g_1M_1+g_2M_2+\cdots+g_nM_n$.
- $g=(g_1, g_2, \cdots, g_n)$ is the encoding vector.
- Encoding can be done recursively.

An Example

- At each node: do linear encoding of the incoming packets.
- $Y=h_1X_1+h_2X_2+h_3X_3$
- Encoding vector is attached with the packet.



Decode

- To recover the original packets M_1, M_2, $\cdots$, M_n.
- Receive m (scrambled) packets.
- How to recover the n unknowns?
  - First, we need $m \geq n$.
  - $m = n$ is sufficient.
- Received packets: Y_1, Y_2, $\cdots$, Y_n.

Coding Scheme

- To decode, we have to solve the linear system:
- $Y_i = a_{i1}M_1 + a_{i2}M_2 + \cdots + a_{in}M_n$
- As long as the coefficients are independent → we can solve the linear system.

- Theorem: (1) There is a deterministic encoding algorithm; (2) Random linear coding is good, with high probability.

Practical Considerations (1)

- Decoding: receiver keeps a decoding matrix recording the packets it received so far.
- When a new packet comes in, its coding vector is inserted at the bottom of the matrix, then perform Gaussian elimination.
- When the matrix is solvable, we are done.

Practical Considerations (2)

- Control the decoding effort (size of the matrix).
- Group packets into generations. Packets in the same generation are encoded.
- Delay: in practice typically not much higher.

Implication of Network Coding

- Successful reception of information
  - does not depend on the receiving packet content.
  - But rather depend on receiving a sufficient number of independent packets.

Summary

- Network coding is good for
  - Scalability
  - Limited topological information
  - Highly dynamic network
- Key insights
  - Treat the packets equally
  - No need to read the content, just do counting
  - Anything helps.
  - Don’t need to know what is where.

Discussion

- More application scenarios?
  - Vehicle network (taxi network)
  - Moving vehicle with access points in proximity.
- New ideas?
- When is network coding is NOT appropriate?

The End