

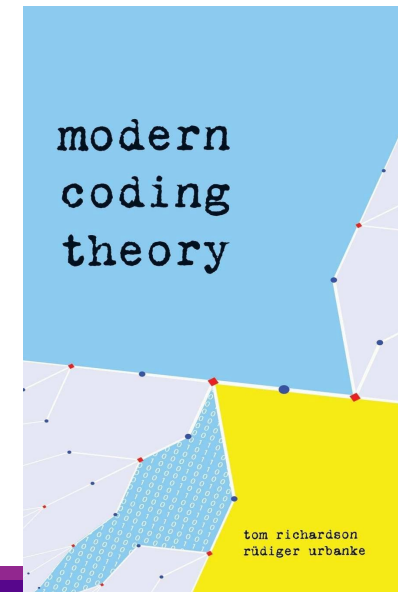
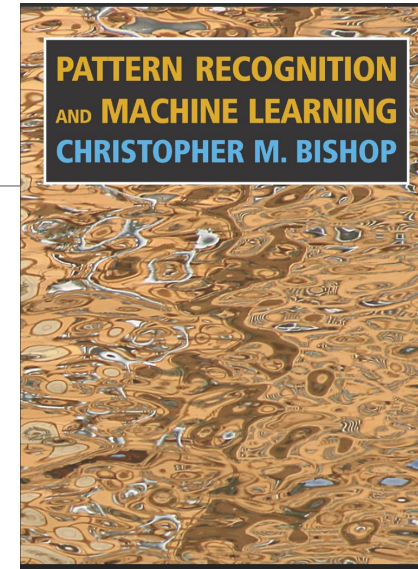
Unit 12: LDPC Codes

EL-GY 6013: DIGITAL COMMUNICATIONS

PROF. SUNDEEP RANGAN

References

- ❑ Extensive tutorial presentation:
 - An Introduction to Low-Density Parity-Check Codes, Paul Siegel
 - https://cmrr-star.ucsd.edu/static/presentations/ldpc_tutorial.pdf
- ❑ Bishop, Pattern Recognition and ML
 - General book on ML from 2006
 - Excellent chapter (Chapter 8) on graphical models
- ❑ Modern Coding Theory, Richardson and Urbanke
 - From 2008
 - By two pioneers in the field
 - Extensive math
- ❑ Richardson, Tom, and Shrinivas Kudekar. "Design of low-density parity check codes for 5G new radio." IEEE Communications Magazine 56.3 (2018): 28-34.



Outline

LDPC Codes: Basics and Motivation

- ☐ LDPC Encoding
- ☐ Graphical Models
- ☐ Inference via Belief Propagation
- ☐ LDPC Decoding
- ☐ 5G LDPC codes

Generator Matrices

□ Consider (n, k) binary linear code

- k information bits $\mathbf{b} = (b_1, \dots, b_k)$
- n coded bits $\mathbf{c} = (c_1, \dots, c_n)$

□ Described by a generator matrix $\mathbf{c} = \mathbf{bG}$

□ Example: (7,4) Hamming Code: $\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$

Parity Check Matrix

□ **Parity check matrix:** An $n \times (n - k)$ matrix H such that $GH = 0$

□ **Properties:**

- For any codeword, \mathbf{c} , $\mathbf{c}H=0$. Why?
- Conversely, if G and H are rank k , then $\mathbf{c}H=0$ implies that \mathbf{c} is a codeword.
[Need some more linear algebra to prove this]

□ For a binary systematic code: $G = [I_k \mid P] \Rightarrow H = \begin{bmatrix} P \\ I_{n-k} \end{bmatrix}$

□ Example: Hamming code $H = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

LDPC Codes

- ❑ Low-Density Parity Check Codes
- ❑ Based on three key ideas
- ❑ Linear codes where the parity check matrix is **sparse**
 - Number of “ones” grows linearly in n
 - Number of “ones” per row is roughly constant
- ❑ **Randomness** in the construction
 - Random placement of “ones”
- ❑ Iterative, **message-passing decoder**
 - Simple “local” decoding at nodes
 - Iterative exchange of information (message-passing)
 - Based on belief propagation from

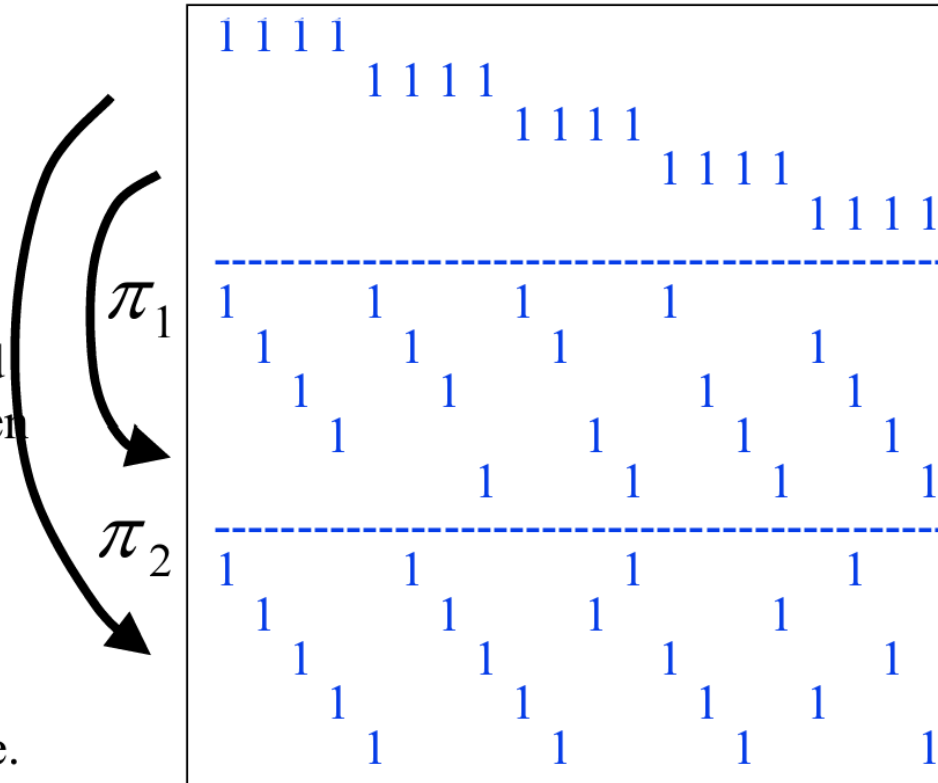
First LDPC codes

- ❑ Gallager, R. G., Low-Density Parity-Check Codes, M.I.T. Press, Cambridge, Mass: 1963.
- ❑ Regular (n, j, k) code
 - n = codeword length
 - Number of 1's in each row of the parity check matrix
 - Number of 1's in each column of the parity check matrix
 - Location of 1's chosen randomly
- ❑ Rate of the code $1 - \frac{j}{k}$
- ❑ Minimum distance tends to grow linearly with n

Gallager's Original Regular Code

$$(n, j, k) = (20, 3, 4)$$

- First $n/k = 5$ rows have $k = 4$ 1's each, descending.
- Next $j-1 = 2$ submatrices of size $n/k \times n = 5 \times 20$ obtained by applying randomly chosen column permutation to first submatrix.
- Result: $jn/k \times n = 15 \times 20$ parity check matrix for a $(n, j, k) = (20, 3, 4)$ LDPC code.



From Siegel, An Introduction to Low-Density Parity-Check Codes

Performance of Regular Codes on BSC

Gallager, 1963

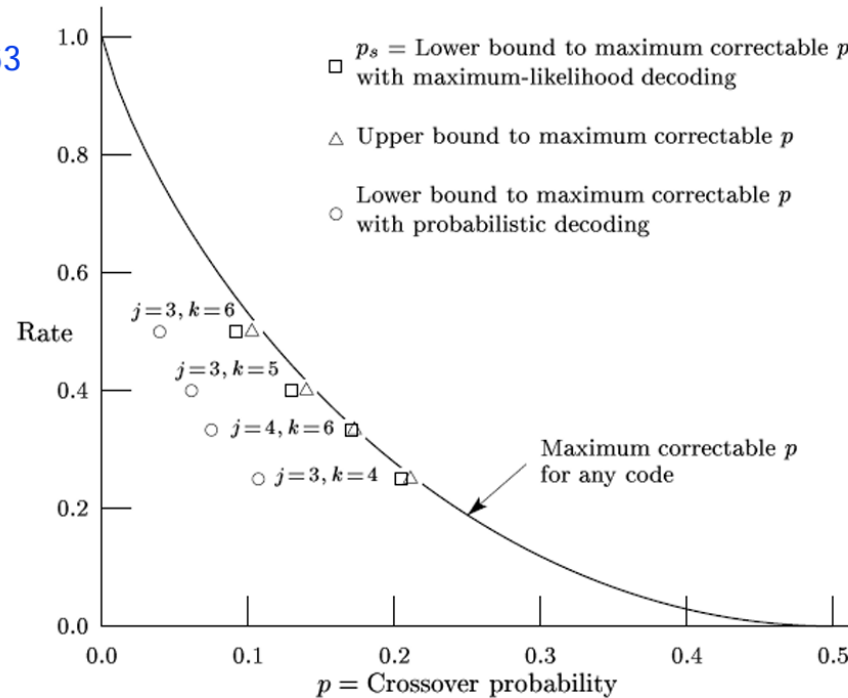
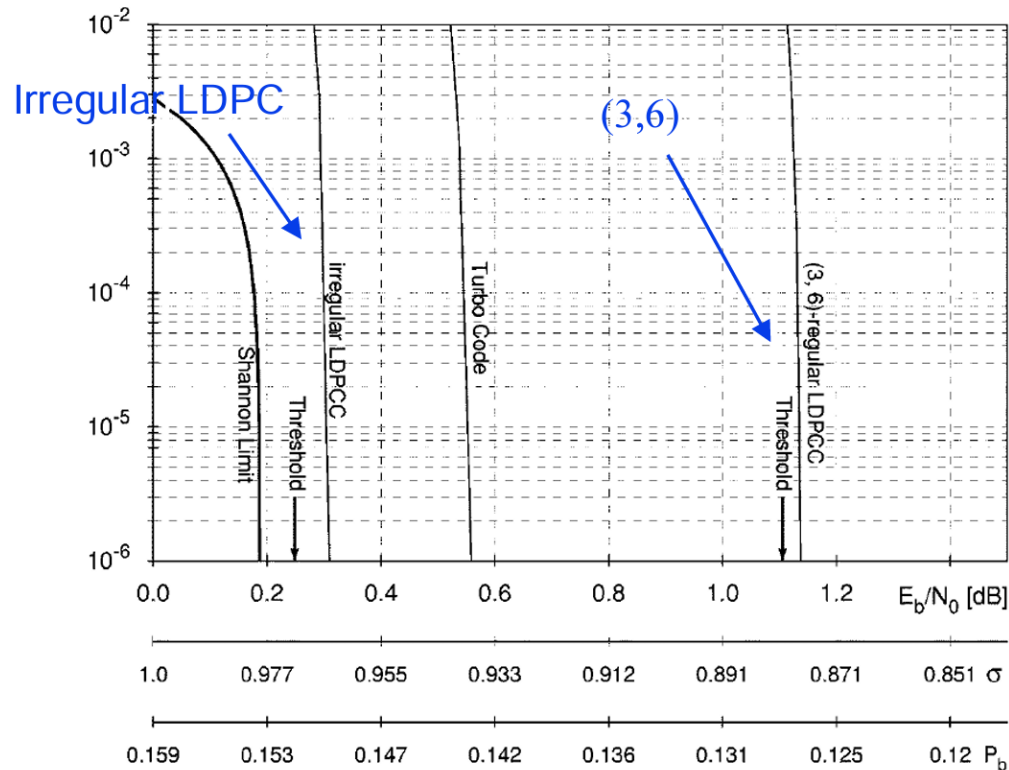


Figure 3.5: Error-correcting properties of (n, j, k) codes on BSC as function of rate for large n .

Enter Irregular Codes



Richardson,
Shokrollahi,
and Urbanke,
2001

$n=10^6$

$R=1/2$

Irregular codes

- Began work in 2001
- Showed significant improvement

On BSC channel at infinite lengths:

- Theoretically hits capacity

At practical code lengths:

- Very close to capacity

Outline

☐ LDPC Codes: Basics and Motivation

 ☒ LDPC Encoding

☐ Graphical Models

☐ Inference via Belief Propagation

☐ LDPC Decoding

☐ 5G LDPC codes

Encoding with a Parity Check Matrix

- ❑ Given a parity check matrix \mathbf{H} how do we encode
- ❑ Parity check matrix can be used to tell if \mathbf{c} is a codeword
- ❑ But does not say how to find a codeword from information bits
- ❑ If $\mathbf{H} = \begin{bmatrix} \mathbf{P} \\ \mathbf{I}_{n-k} \end{bmatrix}$ then we can obtain generator matrix: $\mathbf{G} = [\mathbf{I}_k \mid \mathbf{P}]$
- ❑ Then, we can use \mathbf{G} : $\mathbf{c} = \mathbf{b}\mathbf{G}$
- ❑ But LDPC parity check matrices are not in general systematic

Encoding via Equation Solving

□ Partition code word: $\mathbf{c} = [c_1, \dots, c_{n-k}, c_{n-k+1}, \dots, c_n]$

Parity Info

□ Set last k bits as information bits: $c_{n-k+\ell} = b_\ell$ for $\ell = 1, \dots, k$

□ Write each check node equation:

$$0 = \sum_{\ell=1}^{n-k} c_\ell H_{\ell k} + \sum_{\ell=n-k}^n c_\ell H_{\ell k} \Rightarrow \sum_{\ell=n-k}^n c_\ell H_{\ell k} = \sum_{\ell=1}^{n-k} c_\ell H_{\ell k}$$

- Solve $n - k$ unknowns with $n - k$ equations
- Equations are linear, but over the binary field

Gaussian Elimination

□ Equation solving can be performed via Gaussian elimination

□ Example: Solve: $A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$

◦ Add R1 to R2 and R3: $\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$

◦ Exchange R2 and R3: $\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$

◦ $c_3 = 0$

◦ $c_2 = 1 + c_3 = 1$

◦ $c_1 = c_2 + c_3 = 1$

◦ Solution $c = [1, 1, 0]$

MATLAB

- ❑ MATLAB can also perform binary arithmetic
- ❑ Define arrays with elements in GF(2)
 - The binary field
- ❑ Solve equations just as if they were real numbers

```
% Create the parity check matrix
H = [1,1,1; 1, 1, 0; 1, 0, 1; 0, 1, 1; 1 0 0; 0 1 0; 0 0 1];
H = gf(H);

% Get dimensions
[n,m] = size(H);
k = n-m;

% Generate random bits
b = randi([0,1], 1, k);
b = gf(b);

% Solve c(1:m)H(1:m,:) = b*H(m+1:n,:)
z = b*H(m+1:n,:);
c = z / H(1:m,:);

% Add information bits
c = [c b];
```

Complexity

- In general, we partition the parity check matrix: $\mathbf{H} = \begin{bmatrix} \mathbf{H}^{(1)} \\ \mathbf{H}^{(2)} \end{bmatrix}$
- We solve: $\begin{bmatrix} \mathbf{c}^{(1)} & \mathbf{c}^{(2)} \end{bmatrix} \begin{bmatrix} \mathbf{H}^{(1)} \\ \mathbf{H}^{(2)} \end{bmatrix} = 0 \Rightarrow \mathbf{c}^{(1)} \mathbf{H}^{(1)} = \mathbf{c}^{(2)} \mathbf{H}^{(2)}$
- Since we place information bits in last code bits: $\mathbf{c}^{(2)} = \mathbf{b}$
- Therefore, we must solve: $\mathbf{c}^{(1)} = \mathbf{b} \mathbf{H}^{(2)} (\mathbf{H}^{(1)})^{-1}$
- Complexity: If \mathbf{H} has d elements per column:
 - Multiplication: $\mathbf{b} \mathbf{H}^{(2)}$ has complexity $O(kd)$ ✓
 - $(\mathbf{H}^{(1)})^{-1}$ can be computed once and stored ✓
 - But $(\mathbf{H}^{(1)})^{-1}$ is not, in general, sparse.
 - Multiplication by $(\mathbf{H}^{(1)})^{-1}$ has complexity is $O((n - k)^2)$ ✗


Reducing Complexity

- ❑ For most LDPC codes, complexity can be reduced
- ❑ Find a “mostly” sparse upper triangular $H^{(1)}$
 - Use column permutations with a greedy algorithm
 - Remaining small number of rows that are dense
- ❑ Overall complexity:
 - Pre-processing $O(n^{\frac{3}{2}})$
 - Encoding: $O(n)$

Outline

☐ LDPC Codes: Basics and Motivation

☐ LDPC Encoding

 ☒ Graphical Models

☐ Inference via Belief Propagation

☐ LDPC Decoding

☐ 5G LDPC codes

MAP and Marginal Estimation

□ Given a **posterior distribution**:

$$p(\mathbf{x}|\mathbf{y}), \quad \mathbf{x} = (x_1, \dots, x_N), \quad \mathbf{y} = (y_1, \dots, y_M)$$

- N unknowns: $\mathbf{x} = (x_1, \dots, x_N)$
- M observed variables: $\mathbf{y} = (y_1, \dots, y_M)$

□ Two common estimation problems:

□ **MAP**: Find $\hat{\mathbf{x}} = \arg \max p(\mathbf{x}|\mathbf{y})$

- Finds the most likely unknown vector

□ **Marginal**: Find marginal distribution $p(x_i|\mathbf{y})$ for some i

- Finds the **posterior probability** of a particular variable

Complexity is Generally Exponential

- ❑ Brute force estimation is generally exponentially complex
- ❑ Suppose N unknowns $\mathbf{x} = (x_1, \dots, x_N)$, L choices for each x_i
- ❑ MAP: Find $\hat{\mathbf{x}} = \arg \max p(\mathbf{x}|\mathbf{y})$
 - Search over vectors $\mathbf{x} = (x_1, \dots, x_N)$
 - Search over L^N vectors \mathbf{x}
- ❑ Marginal: For any possible value a for x_i :

$$p(x_i = a|\mathbf{y}) = \sum_{\mathbf{x} : x_i = a} p(\mathbf{x}|\mathbf{y})$$

- Sum is over all vectors \mathbf{x} with $x_i = a$
- For each a , search over L^{N-1} vectors \mathbf{x}

Need structure for tractable estimation

Factorizable Distributions

- Assume posterior distribution is **factorizable**:

$$p(\mathbf{x}|\mathbf{y}) = \frac{1}{Z(\mathbf{y})} \phi_1(\mathbf{x}, \mathbf{y}) \cdots \phi_K(\mathbf{x}, \mathbf{y})$$

- Each term $\phi_k(\mathbf{x}, \mathbf{y})$ is a **factor**

- Assume depends on only a small number, d , of components x_i

- Normalization term $Z(\mathbf{y}) = \sum_{\mathbf{x}} \prod_k \phi_k(\mathbf{x}, \mathbf{y})$

- Ensures posterior density normalizes to one
- Generally, we will not have to explicitly compute $Z(\mathbf{y})$

- **Key idea**: Break hard estimation problem to local problems in each factor

- If d is small, local estimation problems are tractable

Factor Graph

□ Represent statistical relations graphically

□ Factor graph:

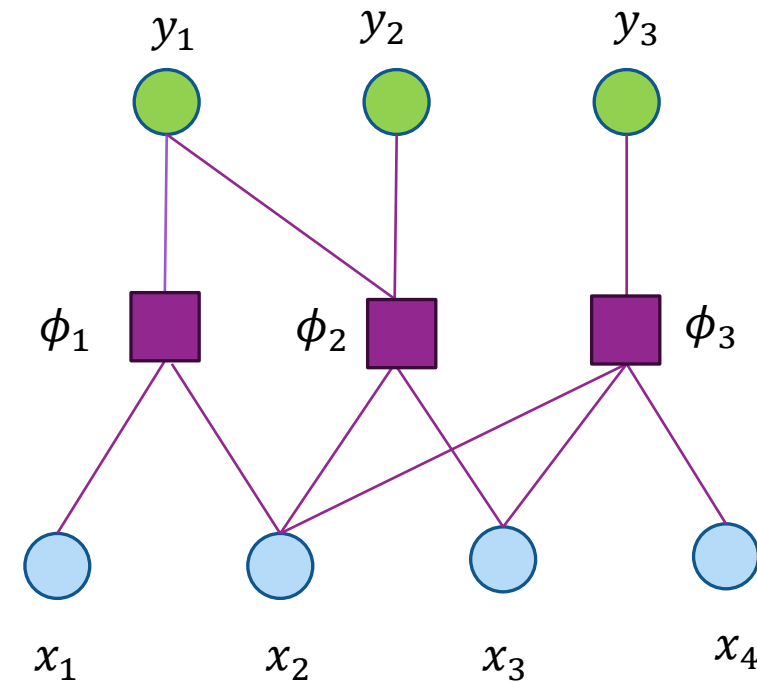
- Unknown variable nodes
- Observed or known variable nodes
- Check or factor nodes

□ Edges between variable and factor nodes

- When variable is part of factor

□ Example:

$$p(x_1, \dots, x_4 | y_1, \dots, y_3) = \frac{1}{Z(y)} \\ \times \phi_1(x_1, x_2, y_1) \phi_2(x_2, x_3, y_1, y_2) \phi_3(x_2, x_3, x_4, y_3)$$



Example: Hidden Markov Chain

□ Markov chain:

- $p(\mathbf{x}) = p(x_0)p(x_1|x_0) \cdots p(x_T|x_{T-1})$

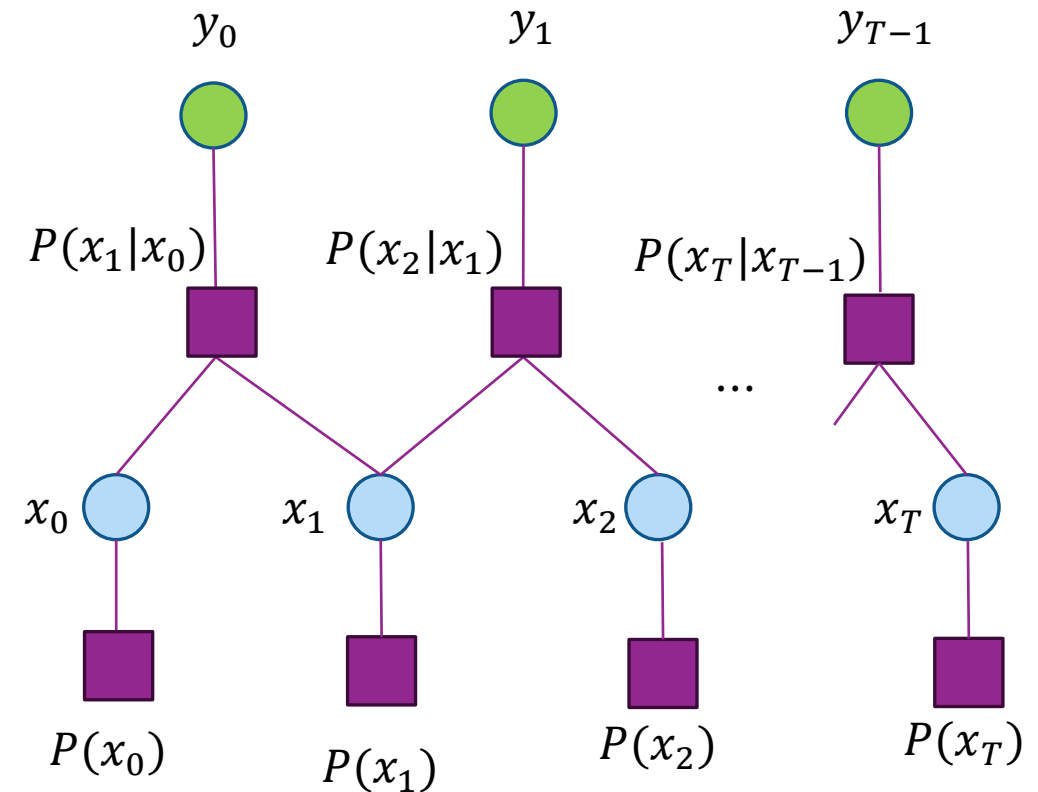
□ Observations

- $p(\mathbf{y}|\mathbf{x}) = p(y_0|x_0) \cdots p(y_{T-1}|x_{T-1})$

□ Hidden Markov chain problem:

- Estimate state sequence $\mathbf{x} = (x_0, \dots, x_T)$
- Use observations $\mathbf{y} = (y_0, \dots, y_{T-1})$

□ Posterior: $p(\mathbf{x}|\mathbf{y}) = \frac{1}{p(\mathbf{y})} p(\mathbf{x})p(\mathbf{y}|\mathbf{x})$



Convolutional Codes

- Example of an HMM

- Consider $1/k$ convolutional code with constraint length K

- State is $x_t = (b_{t-1}, \dots, b_{t-K+1})$ [Last $K - 1$ bits]

- Transition probability:

- Given current state x_t , can transition to one of two new states x_{t+1}

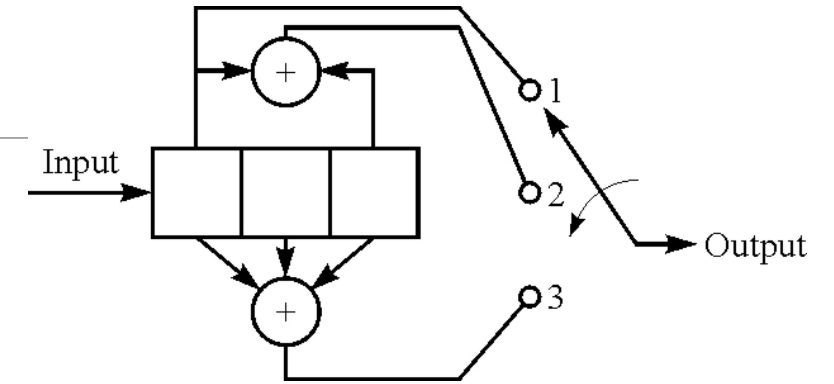
$$P(x_{t+1}|x_t = (b_{t-1}, \dots, b_{t-K+1})) = \begin{cases} 1/2 & x_{t+1} = (?, b_{t-1}, \dots, b_{t-K+2}) \\ 0 & \text{else} \end{cases}$$

- Output is $y_t = (c_{t1}, \dots, c_{tk})$

- Assume bitwise channel $P(r_{ti}|c_{ti})$

- Probability is factorizable:

$$P(r|x) = \prod_{t=0}^{T-1} \prod_{i=1}^k P(r_{ti}|x_t) \times \prod_{t=0}^{T-1} P(x_{t+1}|x_t)$$



Bayes Net Representation

- ❑ Bayes Net: A directed graph
- ❑ Represents conditional probabilities
 - Child is conditioned on parent(s)
 - Example to the right:
$$P(A, \dots, F) = P(A)P(D|A)P(B|A)P(E|A, B)P(C|B, E)P(F|C)$$
- ❑ Can write any Bayes Net as a factor graph
 - Remove directional arrows
 - Add factor node on leaf nodes and edges
- ❑ If a factor graph has no cycles:
it can be represented as Bayes Net

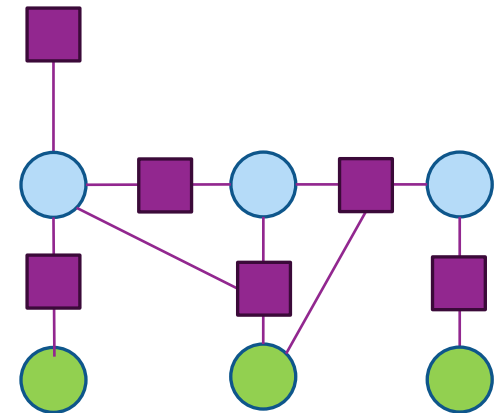
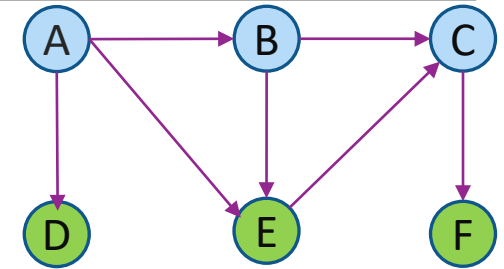


Plate Notation

- Simplify graphical representation
- Write repeated nodes in a “plates”
- Example:
 - HMM with parameter θ
 - $P(x, \theta) = P(x_0)P(\theta)\prod P(x_{t+1}|x_t, \theta)P(y_t|x_t, \theta)$
 - Put update maps in a plate

Standard Representation

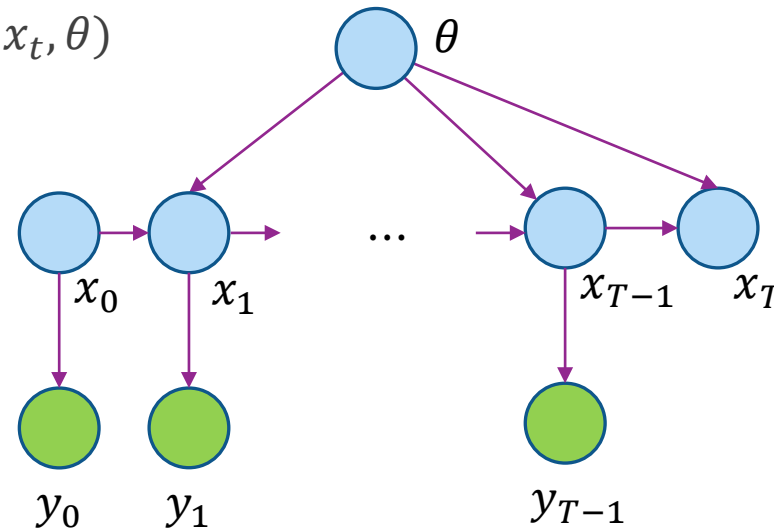
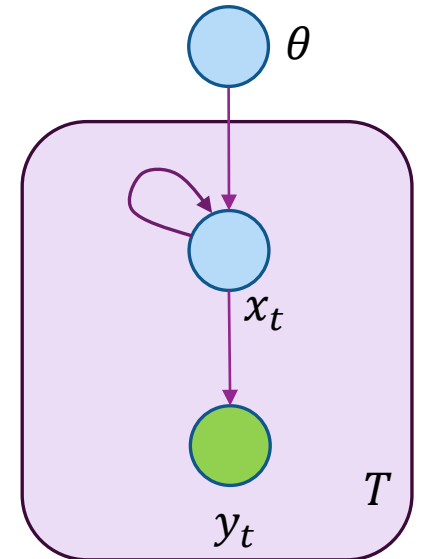
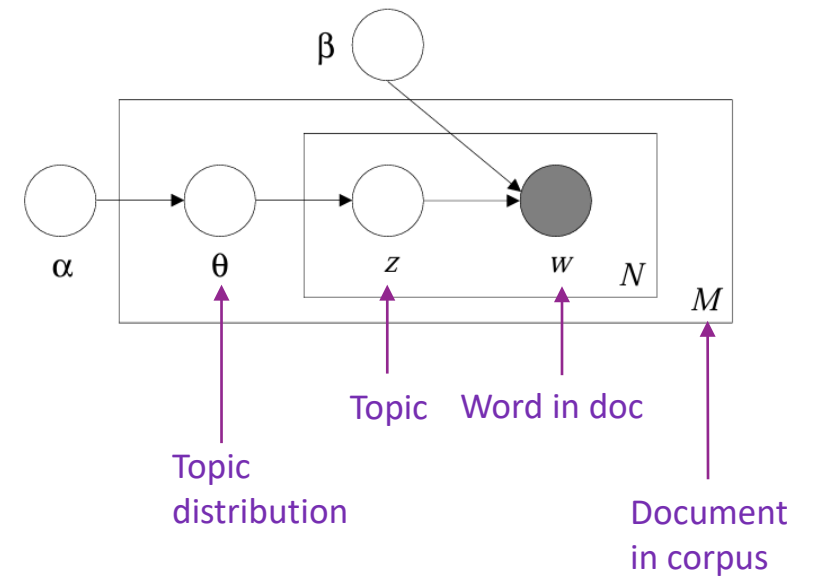


Plate Representation



Examples in ML

- ❑ Graphical model descriptions were used in many ML fields
 - Document modeling
 - Image segmentation
 - ...
- ❑ Allows “explainable” models
 - See example to right
- ❑ Not used much any more
- ❑ Modern deep NN methods outperform graphical models



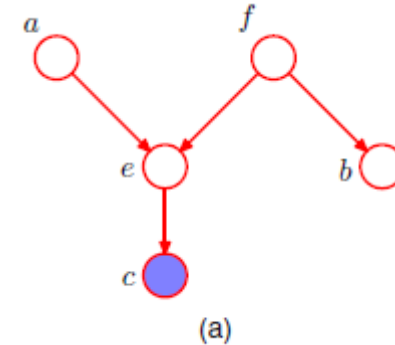
Blei, David M., Andrew Y. Ng, and Michael I. Jordan.
"Latent dirichlet allocation." *Journal of machine Learning research* 3.Jan (2003): 993-1022.

Conditional Independence

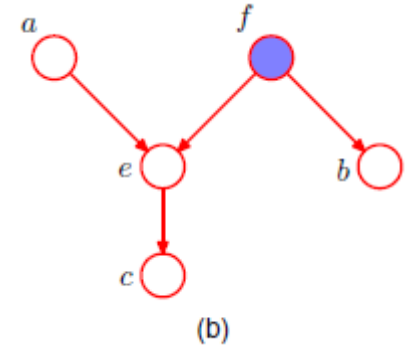
- ❑ Let X, Y and Z be random variables
- ❑ X and Y are independent if: $P(X, Y) = P(X)P(Y)$
- ❑ X and Y are conditionally independent on Z if:
$$P(X, Y|Z) = P(X|Z)P(Y|Z)$$
- ❑ Written $X \perp Y \mid Z$
- ❑ Note:
 - Conditional independence does not \Rightarrow Independence
 - Independence does not \Rightarrow Conditional independence
 -

D-Separation

- ❑ Simple test for a Bayes Net
- ❑ A set of nodes A and B are **d-separated** by node C :
- ❑ For all paths from A to B every node must be:
 - Arrows must tail-tail or head-tail
 - Or, head-head, then the node and its descendants are not in C
- ❑ If A and B are d-separated by C , then A and B are conditionally independent of C



False: $a \perp b \mid c$



True: $a \perp b \mid c$

Example: Time-Varying Noise

□ Model SNR is a Markov chain:

- SNR is time-varying
- $s_t \in \{0, 5, 10\}$ dB. Three possible values
- Transition probability for s_t : $P = \begin{bmatrix} 1-p & p & 0 \\ p/2 & 1-p & p/2 \\ 0 & p & 1-p \end{bmatrix}$
- SNR changes with probability $p = 0.1$ in each time step

□ Transmission:

$$r_t = x_t + w_t, \quad w_t \sim \mathcal{CN}(0, 10^{-0.1s_t})$$

- x_t is a QAM constellation

MATLAB code

```
% Transition probability
Ptrans = [1-p p 0; p/2 1-p p/2; 0 p 1-p];
nstate = length(snrLevel');

% Initial distribution
p0 = [0.4,0.3,0.3];

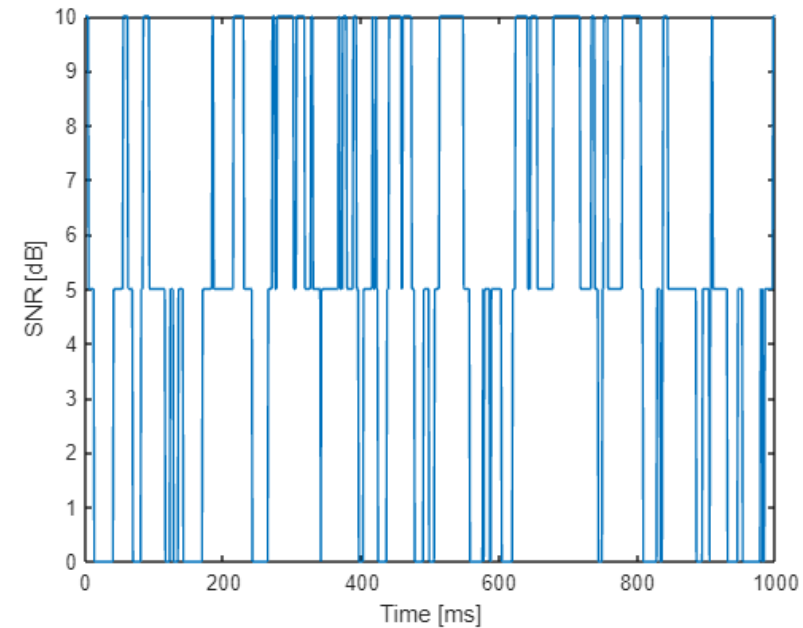
% Get initial state
chanState = zeros(nt+1,1);
state0 = randsample(nstate,1,true,p0);
chanState(1) = state0;

snr = zeros(nt,1);
for t = (1:nt)
    % Get the SNR level based on the current state
    snr(t) = snrLevel(chanState(t));


    % Randomly update state
    p = Ptrans(chanState(t),:);
    chanState(t+1) = randsample(nstate,1,true,p);
end
```

□ Markov chain simulation

- Use randsample function



Outline

- ☐ LDPC Codes: Motivation and History
- ☐ LDPC Encoding
- ☐ Graphical Models
-  ☐ Inference via Belief Propagation
- ☐ LDPC Decoding
- ☐ 5G LDPC codes

Belief Propagation

- ❑ Given a factorizable distribution: $p(\mathbf{x}|\mathbf{y}) = \frac{1}{Z(\mathbf{y})} \phi_1(\mathbf{x}, \mathbf{y}) \cdots \phi_K(\mathbf{x}, \mathbf{y})$
- ❑ **Belief Propagation**: Fast algorithm for:
 - MAP estimation: $\hat{\mathbf{x}} = \arg \max_{\mathbf{x}} p(\mathbf{x}|\mathbf{y})$
 - Marginal distribution: Find $p(x_i|\mathbf{y})$
- ❑ Generalizes the Viterbi algorithm
- ❑ Applies when factor graph is a tree
 - That is, equivalent to a Bayes Net
- ❑ BP complexity is $O(KL^d)$
 - K = number of factors
 - d = max number of unknowns in each factor, L = number of values for each x_i
- ❑ Brute-force complexity is $O(L^{dK})$
 - Significant savings if d is small.
 - Want to break systems in many factors with small number of terms in each factor

BP on a Chain

- Drop dependence on observed variables \mathbf{y}

- Consider a simple factor graph:

$$p(\mathbf{x}) = \frac{1}{Z} \phi_1(x_1) \phi_2(x_1, x_2) \cdots \phi_N(x_{N-1}, x_N)$$

- To compute marginal distribution:

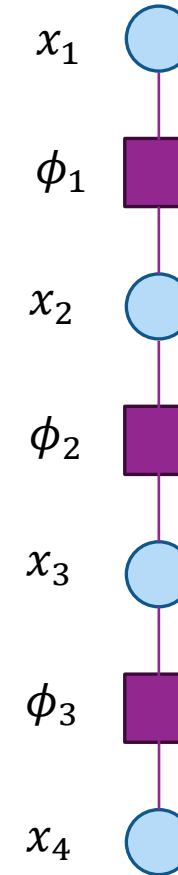
$$p(x_n = j, x_{n-1} = i) = \sum_{\mathbf{x}: x_n = j, x_{n-1} = i} p(\mathbf{x})$$

-

- Sum is over all \mathbf{x} where $x_{n-1} = i, x_n = j$

- Break sum into three terms:

$$p(x_n = i) = \mu_n^+(i)$$



Forward and Reverse Messages

□ Define two partial sums:

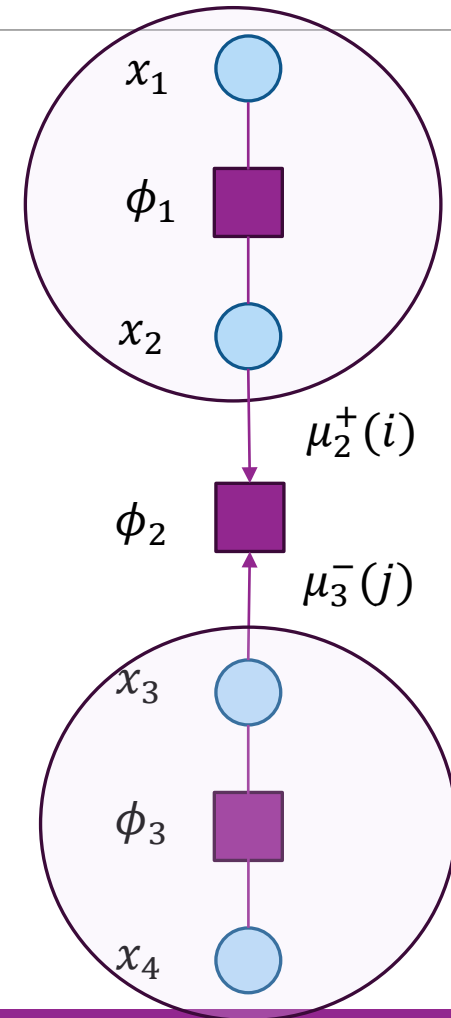
- $\mu_n^+(x_n = i) = \sum_{x_1, \dots, x_{n-1}} \phi_1(x_1) \cdots \phi_{n-1}(x_{n-1}, x_n = i)$
- $\mu_{n+1}^-(x_{n+1} = j) = \sum_{x_{n+2}, \dots, x_N} \phi_{n+1}(x_{n+1} = j, x_{n+2}) \cdots \phi_{N-1}(x_{N-1}, x_N)$

□ Can verify that:

- $P(x_n, x_{n+1}) = \frac{1}{Z} \mu_n^+(x_n) \mu_{n+1}^-(x_{n+1}) \phi(x_n, x_{n+1})$
- $P(x_n) = \frac{1}{Z} \mu_n^+(x_n) \mu_n^-(x_n)$

□ Call:

- $\mu_n^+(x_n)$ the **forward message**
- $\mu_n^-(x_n)$ the **reverse message**



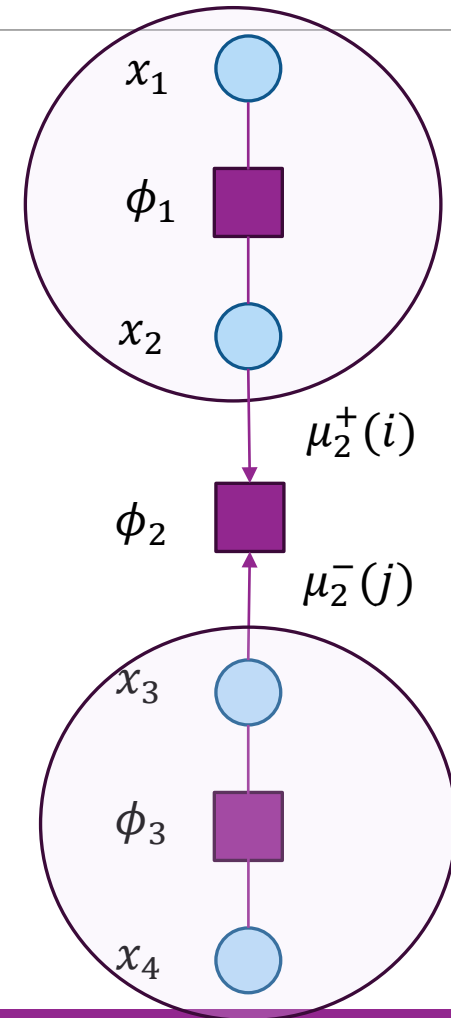
Recursive Updates

□ Forward message:

$$\begin{aligned}\mu_{n+1}^+(x_{n+1}) &= \sum_{x_1, \dots, x_n} \phi_1(x_1) \cdots \phi_{n-1}(x_{n-1}, x_n) \phi_n(x_n, x_{n+1}) \\ &= \sum_{x_n} \phi_n(x_n, x_{n+1}) \sum_{x_1, \dots, x_n} \phi_1(x_1) \cdots \phi_{n-1}(x_{n-1}, x_n) \\ &= \sum_{x_n} \phi_n(x_n, x_{n+1}) \mu_n^+(x_n)\end{aligned}$$

□ Similarly, for the reverse message:

$$\mu_n^-(x_n) = \sum_{x_{n+1}} \phi_n(x_n, x_{n+1}) \mu_{n+1}^-(x_{n+1})$$



Summary: BP on a Chain

- ❑ Compute $\mu_n^+(x_n)$ recursively forward from $n = 1$ to N
- ❑ Compute $\mu_n^-(x_n)$ recursively in reverse from $n = N$ to 1
- ❑ Compute marginal distributions:
 - $P(x_n, x_{n+1}) = \frac{1}{Z} \mu_n^+(x_n) \mu_{n+1}^-(x_{n+1}) \phi(x_n, x_{n+1})$
 - $P(x_n) = \frac{1}{Z} \mu_n^+(x_n) \mu_n^-(x_n)$
- ❑ Note that Z can be found since we know $\sum P(x_n) = 1$
- ❑ Complexity is $O(NL^2)$
 - Linear in time steps N
- ❑ Method is called message passing
 - We pass messages back and forth along the chain

Relation to Bi-LSTMs

□ Bi-LSTMs: Widely-used sequence-to-sequence model in ML

- Input sequence $(x_0, x_1, \dots, x_{T-1})$
- Output sequence $(y_0, y_1, \dots, y_{T-1})$
- Used in NLP, time-series, signal processing, ...

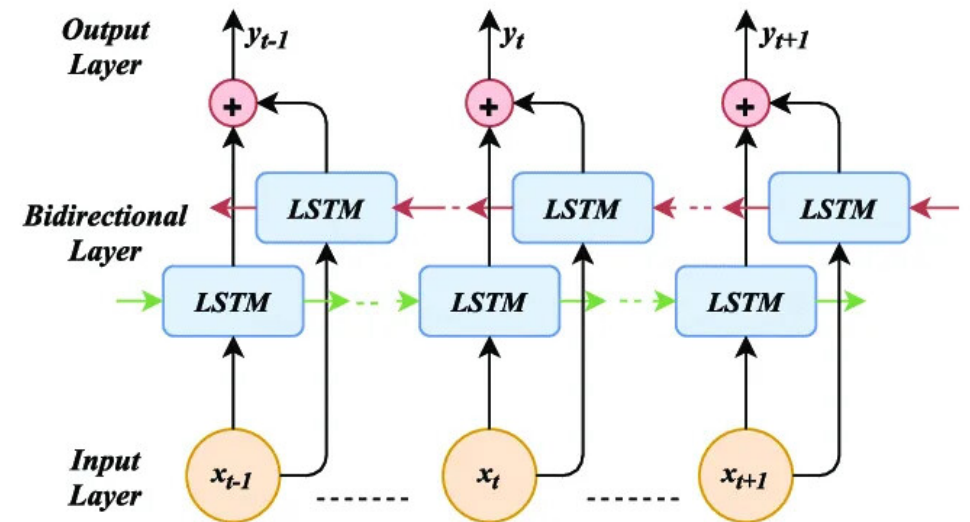
□ Model has a forward and backward structure

□ If the LSTM cell has sufficiently many states

- Can model an optimal state estimator
- x_t is an observation
- y_t is the estimate of state or some function of a state

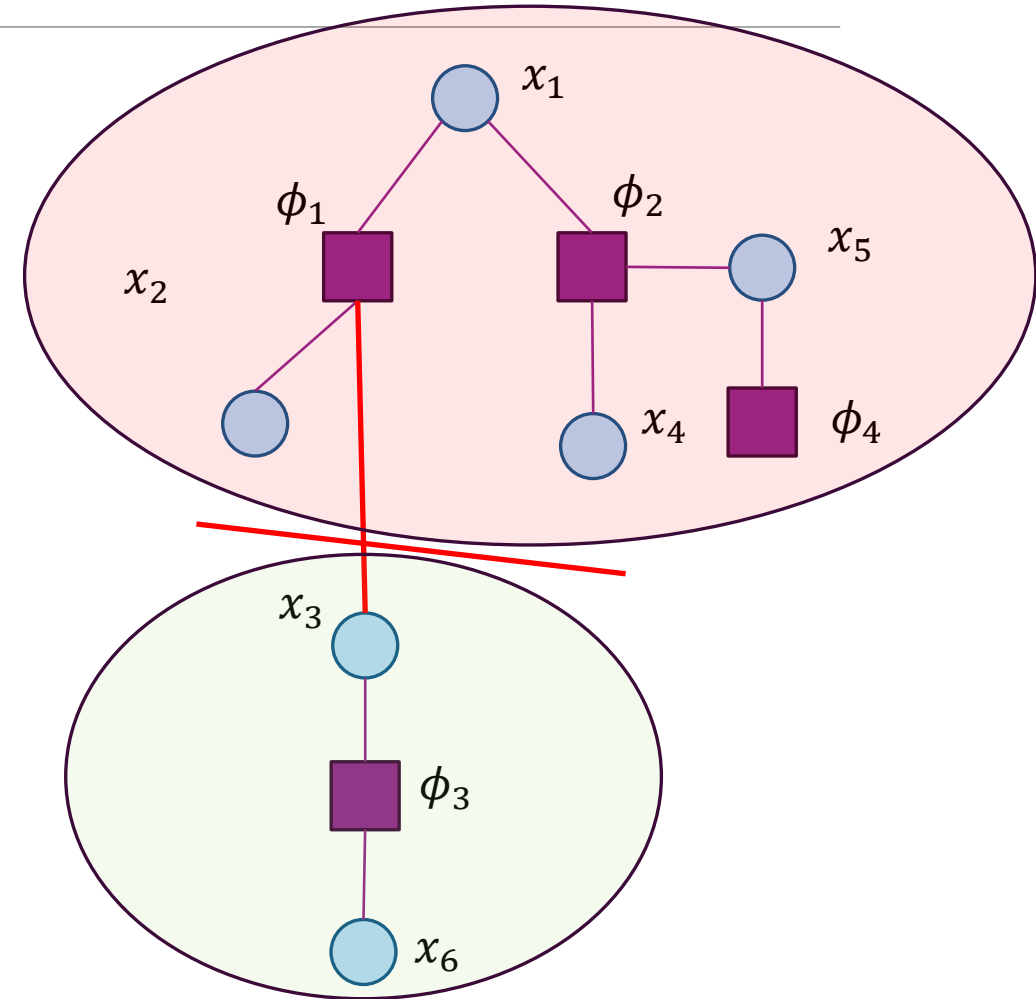
□ LSTMs are trained with data

- Do not need a model of the transition probabilities



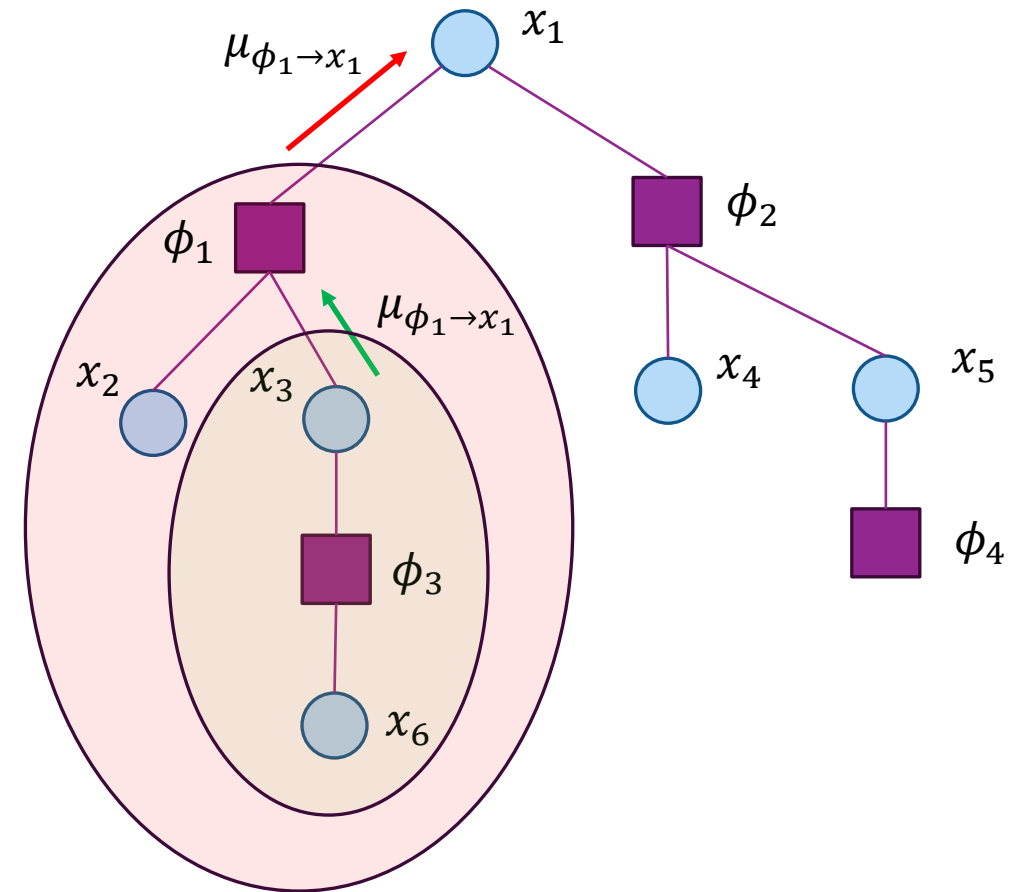
Extensions to a Tree

- BP can be easily extended to **trees**
- Select any node as the **root** of the tree
 - Can be a variable or factor node
- Given any factor ϕ_i define:
 - $x^{(i)}$ = set of variables in factor node i
 - Ex: $x^{(1)} = (x_1, x_2, x_3)$, $x^{(3)} = (x_3, x_6)$
- Given any edge (x_j, ϕ_i) , define:
 - $C^+(x_j, \phi_i) :=$ set of factors on the “root side” of the edge
 - $C^-(x_j, \phi_i) :=$ set of factors on the “leaf side” of the edge
 - Ex: $C^+(x_3, \phi_1) = \{\phi_1, \phi_2, \phi_4\}$, $C^-(x_3, \phi_1) = \{\phi_3\}$



Forward Messages

- Generalize the forward messages on chain:
 - Message towards the tree root
- When x_j is a parent of ϕ_i :
 - $\mu_{\phi_i \rightarrow x_j}(a) = \sum_{x: x_j=a} \prod_{\phi_k \in C^-(\phi_i)} \phi_k(x^{(k)})$
 - Sum is over variables that on the “leaf side” of ϕ_i
 - Ex: $\mu_{\phi_1 \rightarrow x_1}(a) = \sum_{x_2, x_3, x_6} \phi_1(x_1 = a, x_2, x_3) \phi_3(x_3, x_6)$
- When ϕ_i is a parent of x_j
 - $\mu_{x_j \rightarrow \phi_i}(a) = \sum_{x: x_j=a} \prod_{\phi_k \in C^-(x_j)} \phi_k(x^{(k)})$
 - Sum is over variables that on leaf side of x_j
 - Ex: $\mu_{x_3 \rightarrow \phi_1}(a) = \sum_{x_6} \phi_3(x_3 = a, x_6)$



Reverse Messages

□ Generalize the reverse messages on chain:

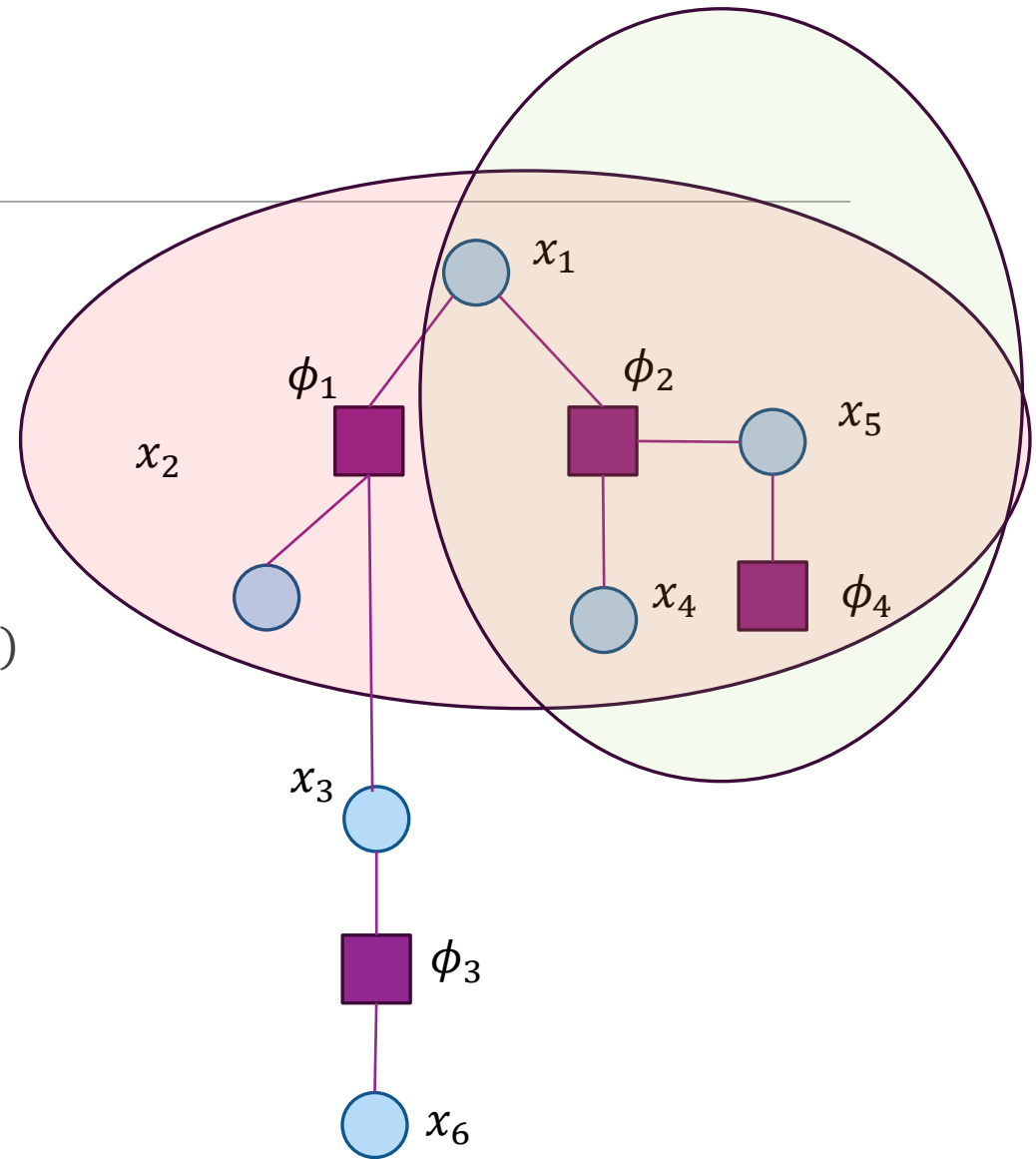
- Message towards the **leaves** of the tree

□ When x_j is a child of ϕ_i :

- $\mu_{\phi_i \rightarrow x_j}(a) = \sum_{x: x_j=a} \prod_{\phi_k \in C^+(\phi_i)} \phi_k(x^{(k)})$
- Sum is over variables on “root” side of ϕ_i
- Ex: $\mu_{\phi_1 \rightarrow x_3}(a) = \sum_{x_1, x_4, x_5} \phi_1(x_1, x_3 = a) \phi_2(x_1, x_4) \phi_4(x_5)$

□ When ϕ_i is a parent of x_j

- $\mu_{x_j \rightarrow \phi_i}(a) = \sum_{x: x_j=a} \prod_{\phi_k \in C^+(x_j)} \phi_k(x^{(k)})$
- Sum is over variables on “root side” of x_j
- Ex: $\mu_{x_1 \rightarrow \phi_1}(a) = \sum_{x_4, x_5} \phi_2(x_1 = a, x_4) \phi_4(x_5)$



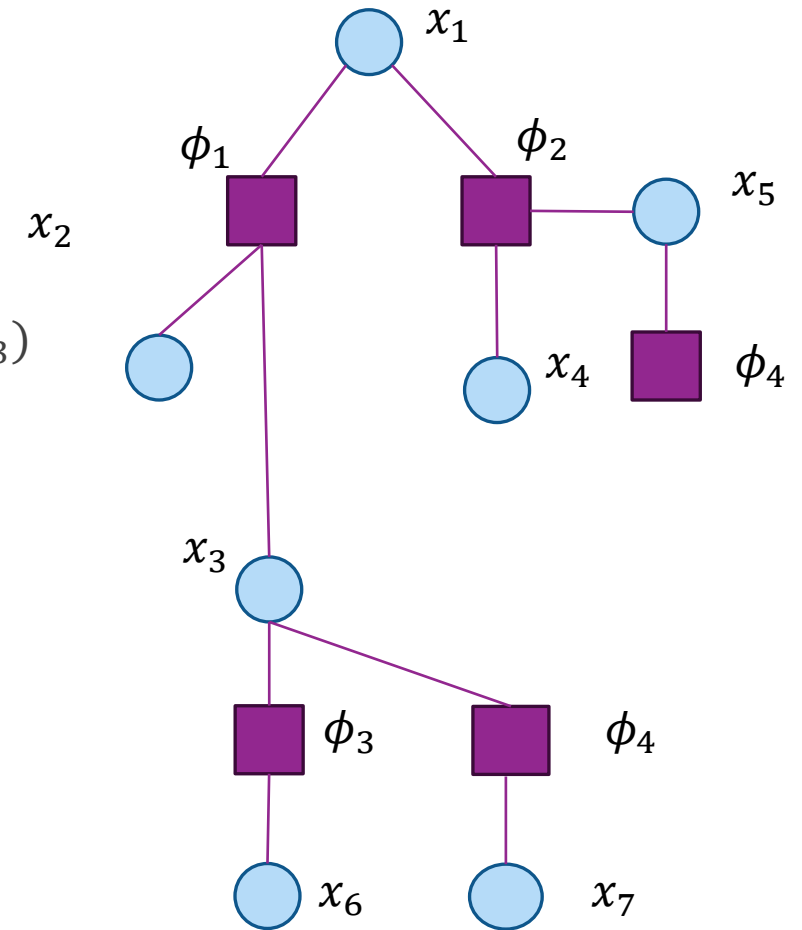
Recursive Updates

□ Messages can be computed recursively

- Let $\partial(\phi_i)$ = variables connected to ϕ_i
- $\mu_{\phi_i \rightarrow x_j}(a) = \sum_{x^{(i)}: x_j=a} \phi_i(x^{(i)}) \prod_{x_k \in \partial(\phi_i) - x_j} \mu_{x_k \rightarrow \phi_i}(x_k)$
- $\mu_{x_j \rightarrow \phi_i}(a) = \prod_{\phi_k \in \partial(x_j) - \phi_i} \mu_{\phi_k \rightarrow x_j}(a)$
- Ex: $\mu_{\phi_1 \rightarrow x_1}(a) = \sum_{x_2, x_3} \phi_1(x_1 = a, x_2, x_3) \mu_{x_2 \rightarrow \phi_1}(x_2) \mu_{x_3 \rightarrow \phi_1}(x_3)$
- Ex: $\mu_{x_3 \rightarrow \phi_1}(a) = \mu_{\phi_3 \rightarrow x_3}(a) \mu_{\phi_4 \rightarrow x_3}(a)$

□ Initialization at leaf nodes:

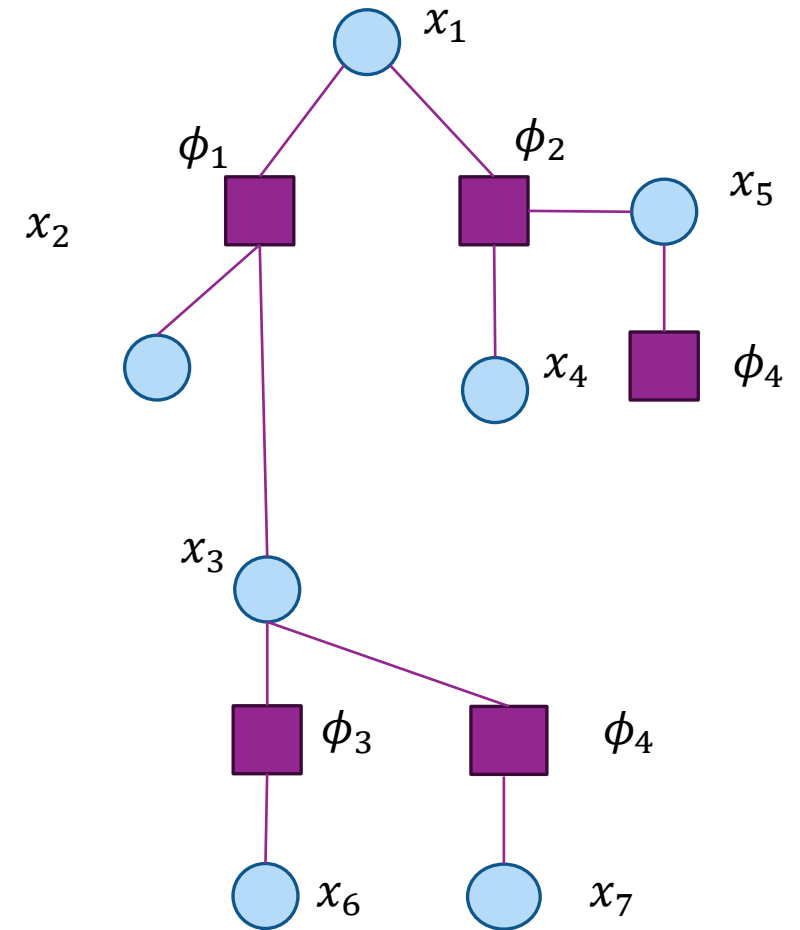
- For variables leaf nodes: $\mu_{x_j \rightarrow \phi_i}(a) = 1$ for all a
- For factor leaf nodes: $\mu_{\phi_i \rightarrow x_j}(a) = \phi_i(x_j = a)$ for all a



BP Algorithm Summary

- ❑ Called the **sum-product algorithm**
- ❑ Select a root node
 - Many nodes may be possible
- ❑ Initial forward messages from leaf nodes
- ❑ Recursively compute messages towards root node
- ❑ Recursively compute all messages back to leaf nodes
- ❑ Final marginal probabilities are:

$$P(x_j) = \frac{1}{Z} \prod_{\phi_k \in \partial(x_j)} \mu_{\phi_k \rightarrow x_j}(a)$$



Max-Sum

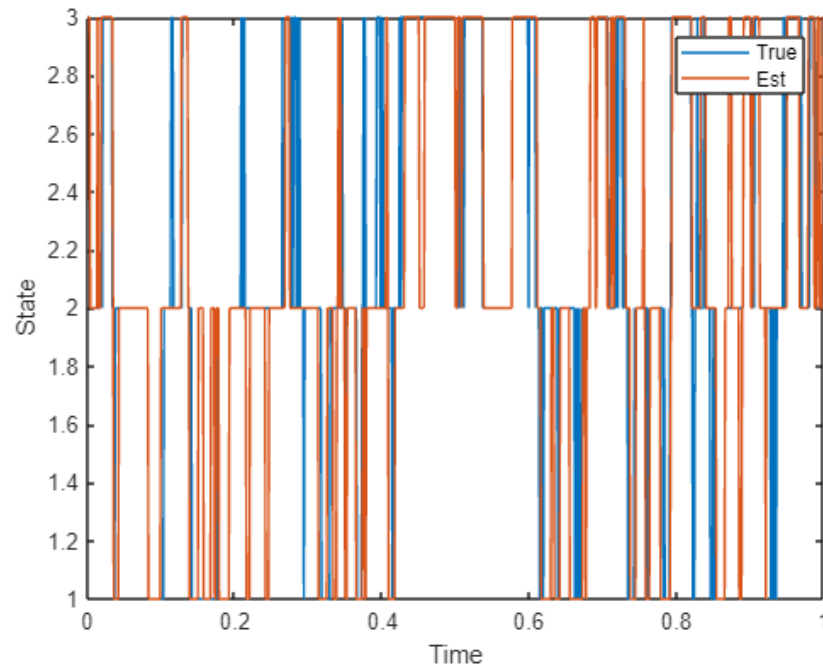
- ❑ **Sum product** computes marginal distributions $P(x_j|\mathbf{y})$
- ❑ **Max-sum** finds MAP estimates: $\hat{\mathbf{x}} = \arg \max p(\mathbf{x}|\mathbf{y})$
 - Replace summations with maximum
- ❑ Example, suppose in sum-product: $\mu_{\phi_1 \rightarrow x_3}(a) = \sum_{x_1, x_4, x_5} \phi_1(x_1, x_3 = a) \phi_2(x_1, x_4) \phi_4(x_5)$
- ❑ In max-sum update is: $\mu_{\phi_1 \rightarrow x_3}(a) = \max_{x_1, x_4, x_5} \phi_1(x_1, x_3 = a) \phi_2(x_1, x_4) \phi_4(x_5)$
- ❑ In a chain, max-sum obtains the Viterbi algorithm
- ❑ See Bishop for more details

Implementation Details

- ❑ Often easier to use $\log \mu_{x \rightarrow \phi}$ instead of $\mu_{x \rightarrow \phi}$.
 - Provides better conditioning
- ❑ Can rescale weights
- ❑ Constant multiplicative factors will not influence result

HMM Example:

- Return to the time varying noise problem
- Simple to implement in MATLAB



```
% Forward pass
mup = p0;
mup(1,:) = ones(1,nstate);
for t=1:nt

    mup(t,:) = mup(t,:).*pr(t,:);
    mup(t,:) = mup(t,:)/ sum(mup(t,:));
    mup(t+1,:) = mup(t,:)*Ptrans;


end

% Reverse pass
mun = zeros(nt+1,nstate);
mun(nt+1,:) = ones(1,nstate);
for t=nt:-1:1
    mun(t,:) = mun(t+1,:)*Ptrans';
    mun(t,:) = mun(t,:).*pr(t,:);
    mun(t,:) = mun(t,:)/ sum(mun(t,:));
end

% Estimated posterior
phat = mun.*mup;
psum = sum(phat,2);
phat = phat ./ psum;

% Compute most likely estimate
[m, im] = max(phat,[],2);
```

Outline

- LDPC Codes: Motivation and History
- Graphical Models
- Inference via Belief Propagation
- LDPC Encoding
-  □ LDPC Decoding
- 5G LDPC codes

Factorizing the Posterior

□ Receive data \mathbf{r}

- Assume bit-wise channel $P(\mathbf{r}|\mathbf{c}) = \prod_{i=1}^n P(r_i|c_i)$

□ Write parity check equations $\mathbf{z} = \mathbf{cH}$

- Let $N_i = \{j | H_{ij} = 1\}$
- Each check node is a linear equation $z_i = \sum_j H_{ij}c_j = \sum_{j \in N_i} c_j$
- We need $z_i = 0$ for all check nodes i
- Write a penalty function $\phi_i(z_i) = \begin{cases} 1 & z_i = 0 \\ 0 & z_i = 1 \end{cases}$
- Penalty $\phi_i(z_i) = \phi_i(c^{(i)})$, is a function of the neighbors of z_i : $c^{(i)} = \{c_j, j \in N_i\}$

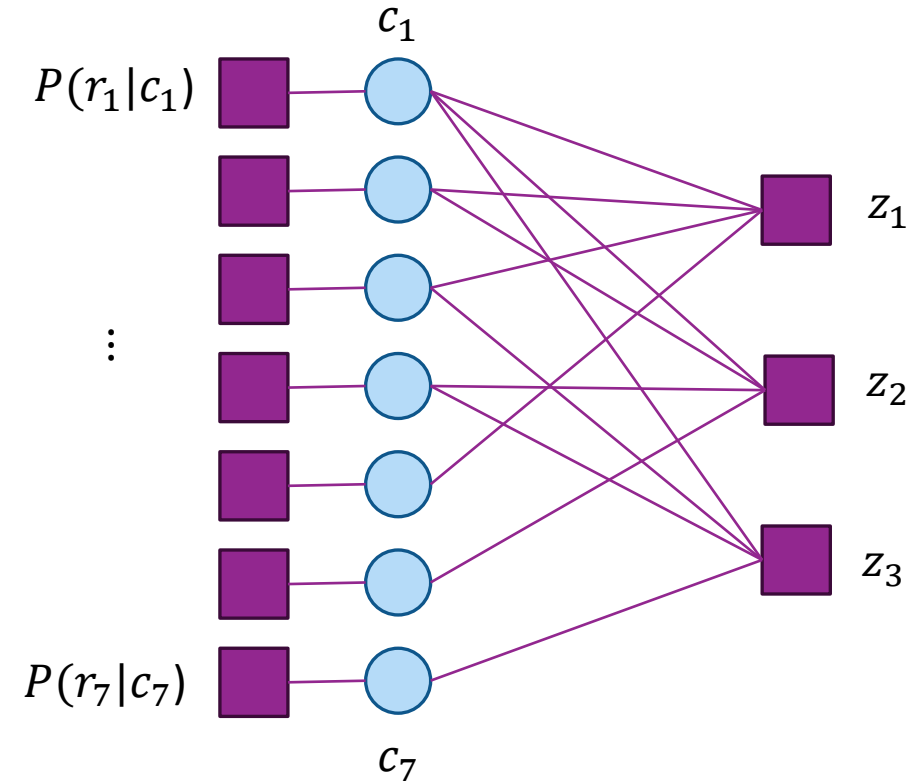
□ Posterior probability has a factorizable form:

$$P(\mathbf{c}|\mathbf{r}) \propto P(\mathbf{r}|\mathbf{c})P(\mathbf{c}) \propto \prod_{i=1}^{n-k} \phi_i(c^{(i)}) \prod_{j=1}^n P(r_j|c_j)$$

Factor Graph

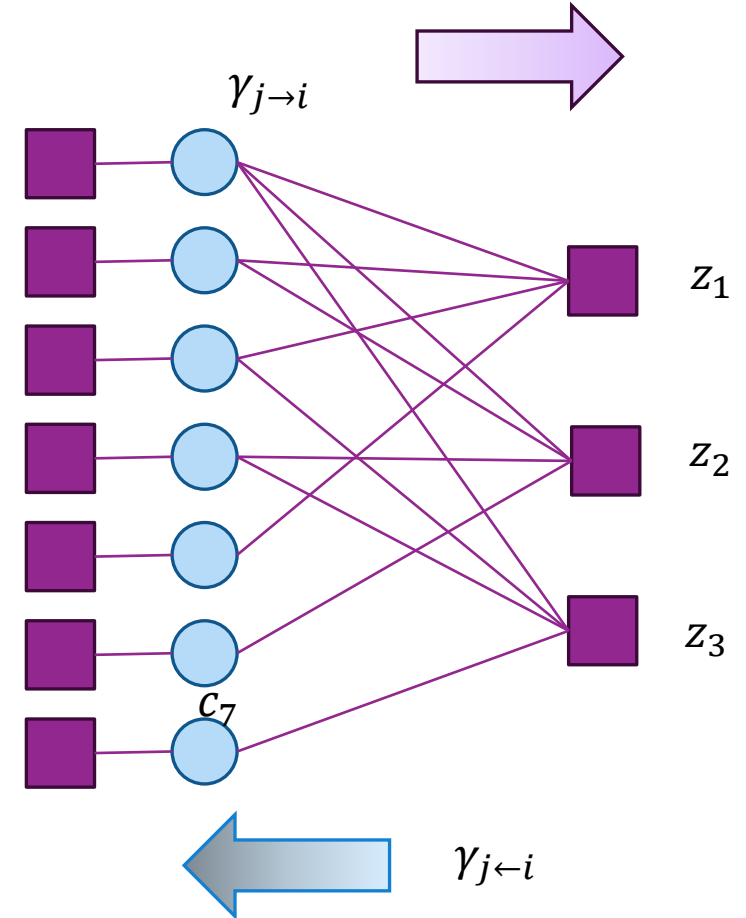
- Also called the **Tanner Graph**
- Variable nodes:
 - One for each coded bit c_j
- Factor nodes
 - One for each $P(r_j|c_j)$ [These are often not shown]
 - One for each check node z_i
- Example: Hamming

- $H = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$



Message Passing Decoding with LLRs

- ❑ Use LLRs instead of messages
 - Each variable has two values $c_j = 0$ or 1
- ❑ Easier to write updates with LLRs
- ❑ Variable to check node message:
 - $\gamma_{j \rightarrow i} := \log \frac{\mu_{j \rightarrow i}(c_j=1)}{\mu_{j \rightarrow i}(c_j=0)}$
- ❑ Check to variable messages:
 - $\gamma_{j \leftarrow i} := \log \frac{\mu_{j \leftarrow i}(c_j=1)}{\mu_{j \leftarrow i}(c_j=0)}$



Variable Node Update

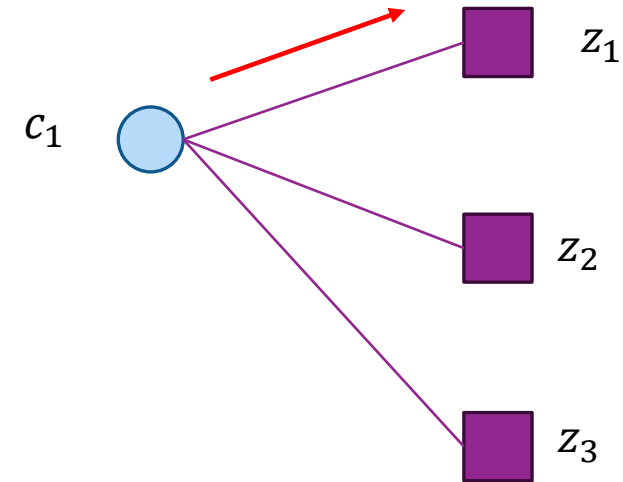
□ Consider message from some c_j to z_i

□ LLR derivation:

- $\mu_{j \rightarrow i}(c_j = 1) = \prod_{z_k \in N(c_j) - z_i} \mu_{j \leftarrow k}(c_j = 1)$
- $\mu_{j \rightarrow i}(c_j = 0) = \prod_{z_k \in N(c_j) - z_i} \mu_{j \leftarrow k}(c_j = 0)$
- $\gamma_{j \rightarrow i} = \log \left[\prod_{z_k \in N(c_j) - z_i} \frac{\mu_{j \leftarrow k}(c_j = 1)}{\mu_{j \leftarrow k}(c_j = 0)} \right]$
 $= \sum_{z_k \in N(c_j) - z_i} \gamma_{j \leftarrow k}$

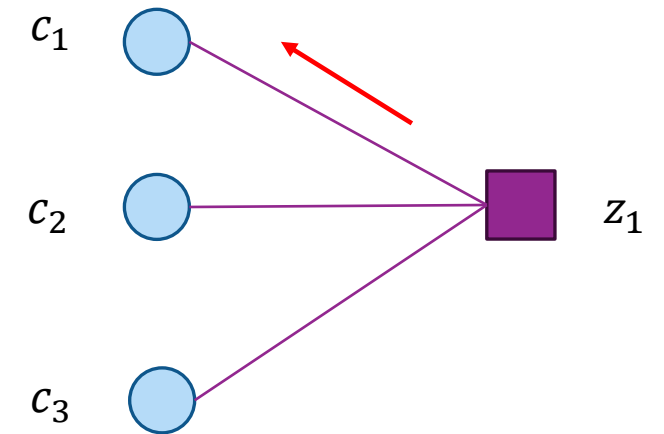
□ Example in graph to the right:

- $\gamma_{1 \rightarrow 1} = \gamma_{1 \leftarrow 2} + \gamma_{1 \leftarrow 3}$
- $\gamma_{1 \rightarrow 2} = \gamma_{1 \leftarrow 1} + \gamma_{1 \leftarrow 3}$



Factor Node Update 1

- Consider message from z_i to c_j
- We know $z_i = \sum_{k \in N_i} c_k = 0$
- Define:
 - $A_0 = \{ c^{(i)} \mid \sum_{k \in N_i} c_k = 0, c_j = 0 \},$
 - $A_1 = \{ c^{(i)} \mid \sum_{k \in N_i} c_k = 0, c_j = 1 \},$
- Example, in message from z_1 to c_1 in graph to right:
 - We want $c_1 + c_2 + c_3 = 0$
 - $A_0 = \{ (c_2, c_3) = (0,0), (1,1) \}$
 - $A_1 = \{ (c_2, c_3) = (1,0), (0,1) \}$



Factor Node Update 2

□ Then $\mu_{j \leftarrow i}(c_j = 0) = \sum_{c^{(i)} \in A_0} \prod_{c_k} \mu_{k \rightarrow i}(c_k)$

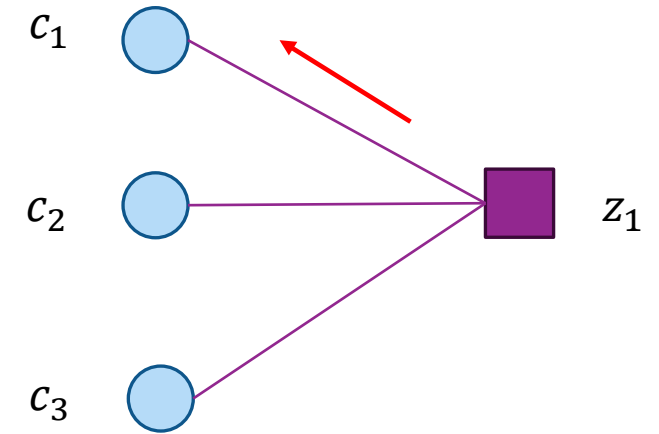
□ Since $\gamma_{j \rightarrow i} := \log \frac{\mu_{j \rightarrow i}(c_j=1)}{\mu_{j \rightarrow i}(c_j=0)}$

- $\mu_{j \leftarrow i}(c_j = 0) = B \sum_{c^{(i)} \in A_0} \exp[\sum c_k \gamma_{k \rightarrow i}]$
- $B = \prod_k \mu_{k \rightarrow i}(0)$

□ Similarly, $\mu_{j \leftarrow i}(c_j = 1) = B \sum_{c^{(i)} \in A_1} \exp[\sum c_k \gamma_{k \rightarrow i}]$


□ LLR message is:

$$\begin{aligned} \gamma_{i \rightarrow j} &:= \log \frac{\mu_{i \rightarrow j}(c_j=1)}{\mu_{i \rightarrow j}(c_j=0)} \\ &= \log \left[\frac{\sum_{c^{(i)} \in A_1} \exp[\sum c_k \gamma_{k \rightarrow i}]}{\sum_{c^{(i)} \in A_0} \exp[\sum c_k \gamma_{k \rightarrow i}]} \right] \end{aligned}$$



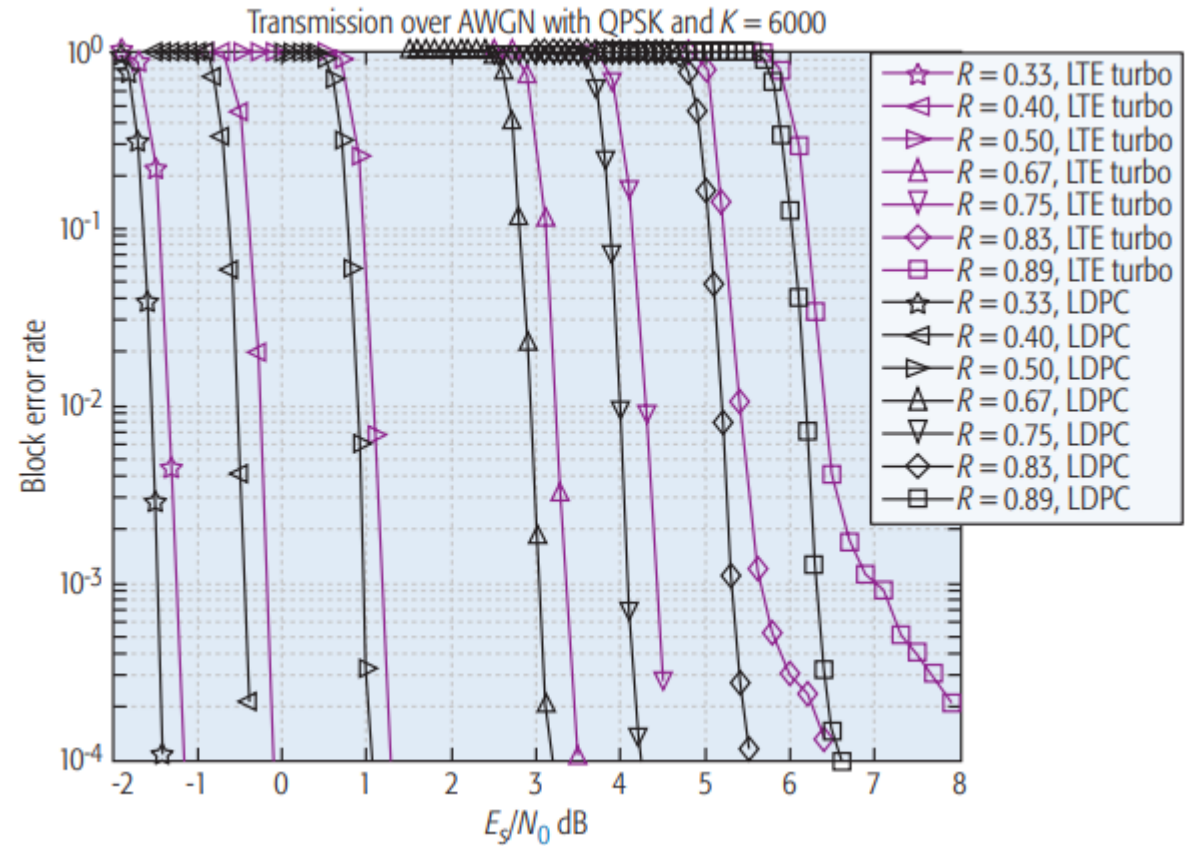
$$\gamma_{1 \rightarrow 1} = \log \left[\frac{e^{\gamma_{2 \rightarrow 1}} + e^{\gamma_{3 \rightarrow 1}}}{1 + e^{\gamma_{3 \rightarrow 1} + \gamma_{2 \rightarrow 1}}} \right]$$

Outline

- LDPC Codes: Motivation and History
- Graphical Models
- Inference via Belief Propagation
- LDPC Encoding
- LDPC Decoding
-  □ 5G LDPC codes

LDPC vs. Turbo

- Move for Turbo (4G) to LDPC (5G)
- Slight performance improvement
- Complexity is lower



Structure

- ❑ Base graph:
 - A small LDPC graph
 - Two base graphs supported in 5G
- ❑ Lifting:
 - Used to create longer block lengths
- ❑ Number of columns and rows adjustable:
 - To support different rates
 - Support incremental redundancy
- ❑ We will discuss IR in the wireless class

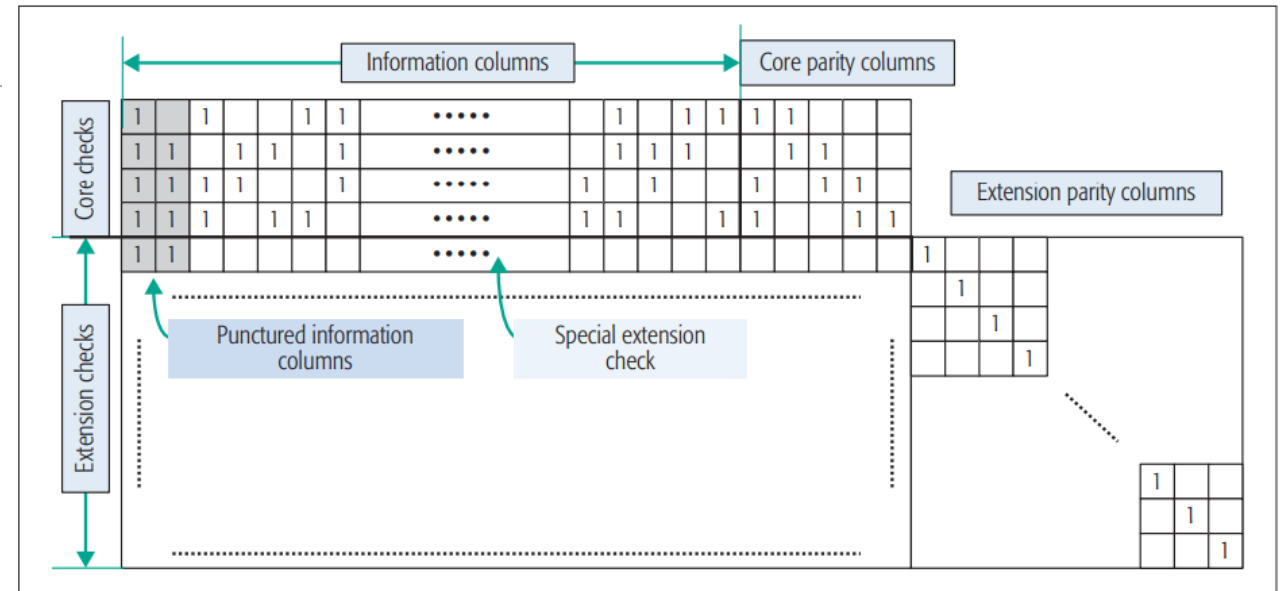


Figure 5. Sketch of base parity check structure for the 5G NR LDPC code.