# An Introduction to Low-Density Parity Check Codes

Daniel J. Costello, Jr.
Department of Electrical Engineering
University of Notre Dame

August 10, 2009

The author gratefully acknowledges the help of Tom Fuja
in the preparation of this presentation.

# Things You Already Know

## Some Basic Definitions:

- An $(n, k)$ linear binary block code $\mathcal{C}$ is a $k$-dimensional subspace of $\{0, 1\}^n$.
- A *generator matrix* for $\mathcal{C}$ is a binary matrix $G$ whose rows span $\mathcal{C}$. (So $G$ is an $\ell \times n$ matrix, where $\ell \geq k$.)
- A parity check matrix for $\mathcal{C}$ is a binary matrix $H$ whose rows span $\mathcal{C}^{\perp}$ - i.e., $\mathbf{c} \in \mathcal{C}$ if and only if $\mathbf{c}H^T = \mathbf{0}$. (So $H$ is an $m \times n$ binary matrix, where $m \geq n - k$.)

# Gallager's Early Work

**Definition:** A low-density parity check (LDPC) code is a linear binary block code for which the parity check matrix of interest has a low density of ones. (Gallager, 1962)

**So:**

- LDPC really refers to a *representation* of a code rather than the code itself.
  - A code may have one representation that is low-density and another that is *not* low-density.
  - By referring to "LDPC codes" we refer to codes with a low-density representation - and we will exploit that representation.
- The word "low" is a vague term.
  - We shall see that decoding complexity increases with the density of ones - so it's in our interest to keep that density low.

# Gallager's Early Work, continued

**Definition:** A *regular* LDPC code is one for which the $m \times n$ parity check matrix of interest has $w_c$ one's in every column and $w_r$ ones in every row.

- Each code bit is involved with $w_c$ parity constraints and each parity constraint involves $w_r$ bits.
- "Low density" means $w_c \ll m$ and $w_r \ll n$.
- $w_c n = w_r m =$ number of ones in $H$.
- So $m \geq n - k$ means $R = k/n \geq 1 - (w_c/w_r)$, and thus $w_c < w_r$.
- We sometimes refer to such a code as $(w_c, w_r)$ regular.

**Definition:** An *irregular* LDPC code in which the row weights and/or column weights of the parity check matrix are not constant.

# Gallager's Early Work, continued

**An Example:** Consider the parity check matrix for the (2,4) regular LDPC code given below:

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

$$n = 10 \qquad m = 5 \qquad n - k = 4 \qquad w_c = 2 \qquad w_r = 4$$

# Gallager's Early Work, continued

**Fact:** Regular LDPC codes are "asymptotically good".

- This means that there exists a sequence of regular LDPC codes with increasing blocklength $n$ such that

$$\limsup_{n\to\infty} \frac{k}{n} > 0 \quad \text{and} \quad \limsup_{n\to\infty} \frac{d_{\min,n}}{n} > 0.$$

- Many classes of codes - e.g., BCH codes - are *not* asymptotically good.

# Gallager's Early Work, continued

## More specifically:

- In his Ph.D. thesis, Gallager showed that the minimum distance of a randomly chosen code selected from a particular ensemble of regular LDPC codes has a cumulative distribution function that looks like this as $n$ gets larger:

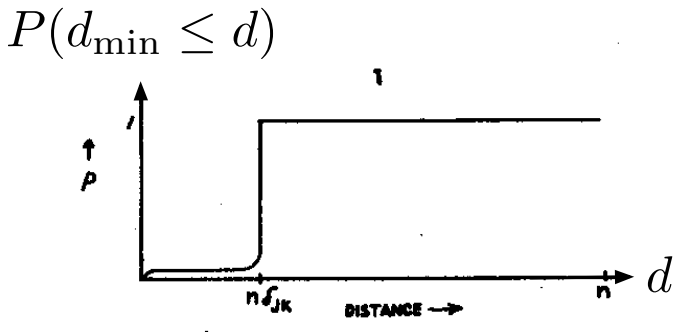
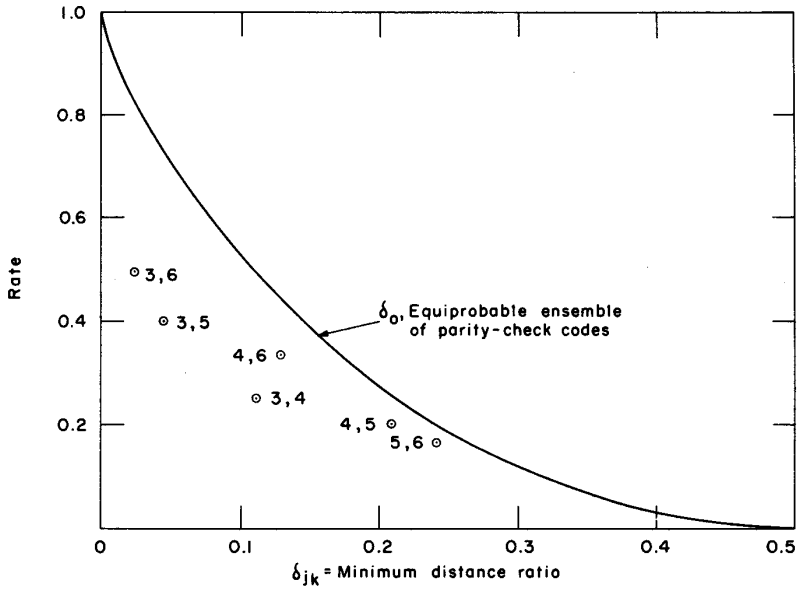



Figure 1: Cumulative distribution function of the minimum distance of a randomly selected (j,k) regular LDPC code.

- Gallager's results assume $w_r > w_c \geq 3$;
- The fractional weight $\delta_{j,k}$ depends on the column weight $w_c = j$ and row weight $w_r = k$.

# Gallager's Early Work, continued

- Example values of $\delta_{w_c,w_r}$ are plotted below.
- Also graphed is $\delta_0$ - a probabilistic bound for conventional (non-low-density) codes that is equivalent to the Gilbert-Varshamov bound.



# Gallager's Early Work, continued

## Hard-Decision Bit Flipping Decoding - A Simple Example

1. Fix a "threshold" parameter $\delta$. ($\delta$ can be optimized.)
2. For parity check $j$ $(0 \leq j \leq m-1)$, compute the associated syndrome $S_j$.
3. If $S_j = 0$ for all $j$ or you've completed a maximum number of iterations, stop.
4. For each bit position $i$ $(0 \leq i \leq n-1)$, let $g_i$ denote the number of non-zero syndromes that include bit $i$.
5. Let $\mathcal{A}$ denote the bit positions that participate in more than $\delta$ failed parity checks - i.e., $\mathcal{A} = \{i : g_i > \delta\}$.
6. Flip bit $i$ for all $i \in \mathcal{A}$ and go to Step 2.

*Note:* If the code is code is $(w_c, w_r)$-regular, then each parity check is affected by at most $w_r$ code bits and $g_i \leq w_c$.

# Tanner Graphs

**Definitions:**

- A *bipartite graph* is one in which the nodes can be partitioned into two classes, and no edge connects two nodes from the same class.
- A *Tanner graph* for an LDPC code is a bipartite graph such that:
  - In the first class of nodes, there is one node for each of the $n$ bits in the codeword - i.e., the "bit nodes" or the "variable nodes."
  - In the second class of nodes, there is one node for each of the $m$ parity checks - i.e., the "check nodes" or the "function nodes."
  - An edge connects a bit node to a check node if and only if the bit is included in the parity check.

# Tanner Graphs, continued

**Example:** Consider the parity check matrix and Tanner graph shown below.

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

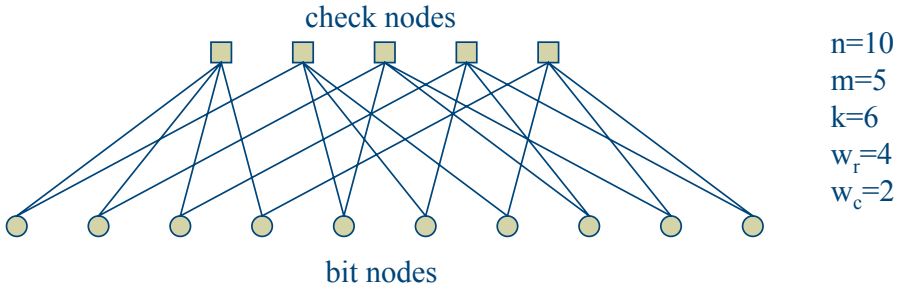



Figure 2: Example of a Tanner graph.

# Tanner Graphs, continued

**Definition:** A *cycle* of length $\ell$ in a Tanner graph is a path comprised of $\ell$ edges from a node back to the same node.

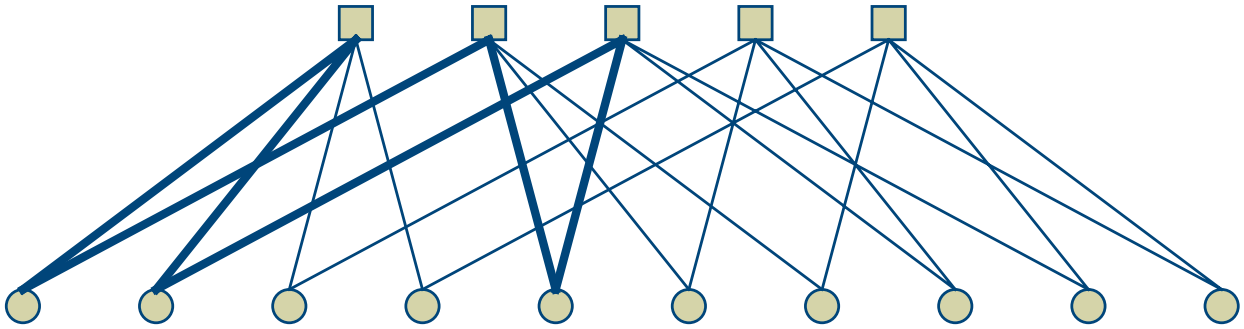


Figure 3: A cycle of length six in a Tanner graph.

# Tanner Graphs, continued

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

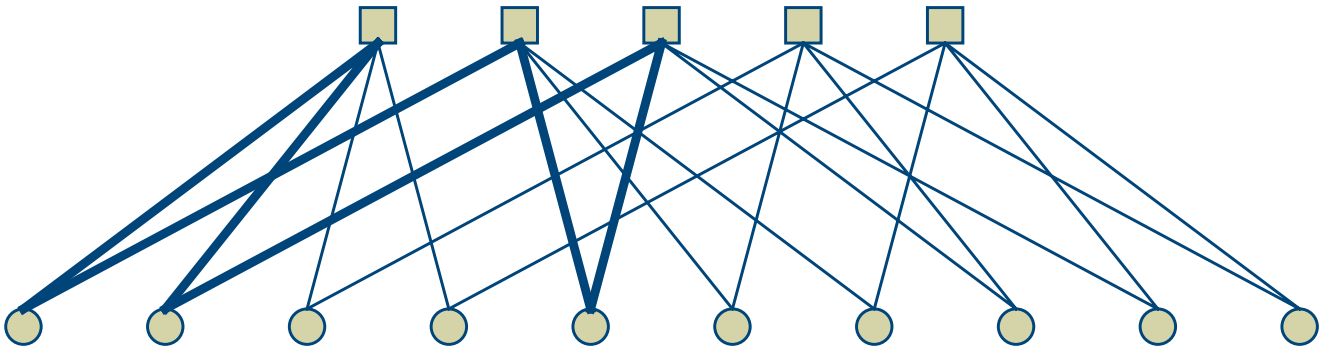


Figure 4: A cycle of length six as seen in both the Tanner graph and the parity check matrix.

# Tanner Graphs, continued

**Definition:** The *girth* of a Tanner graph is the minimum length of any cycle in the graph.

- Obviously, the shortest possible cycle in any Tanner graph is four. (Indicated in $H$ by a "rectangle" of four 1's.)
  - Note: Some define LDPC codes so that graphs with 4-cycles are not included (e.g., Lin/Costello, Def. 17.1).
- The girth of our example is six.
- Short cycles are usually considered bad in graphs used for iterative decoding based on "message passing."
  - Short cycles increase the dependence of information being received at each node during message passing.

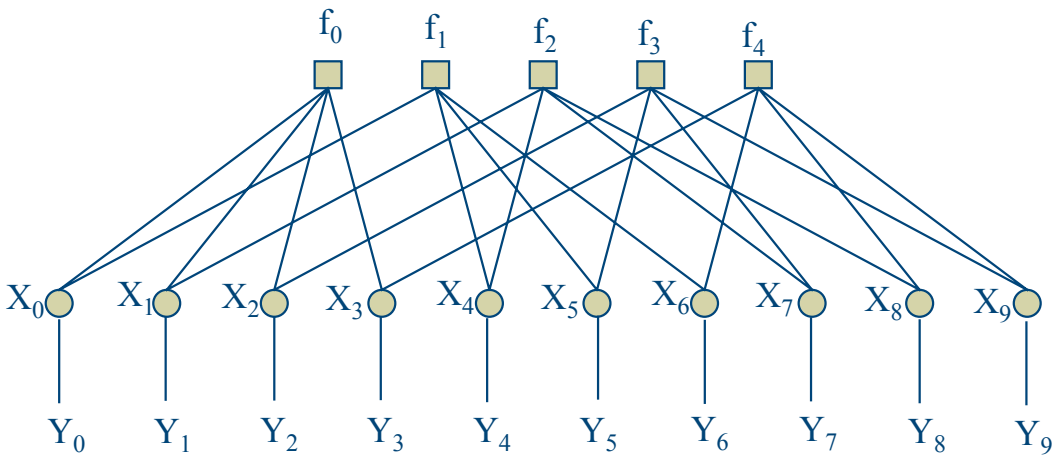
# Tanner Graphs, continued

– Analysis of message passing usually assumes independent information arriving at each node.

– But Shu Lin (and others) have constructed LDPC codes based on graphs with short cycles that perform extremely well under iterative decoding.

# Tanner Graphs, continued

- For decoding purposes, it's convenient to modify the Tanner graph as shown:




Here, $Y_i = X_i + n_i$, where $X_i = (-1)^{c_i} \in \{+1, -1\}$ and $n_i$ is Gaussian with mean zero and variance $\sigma^2$. ($c_i \in \{0, 1\}$)

# Message Passing

## Big Picture:

- Message passing - a.k.a. belief propagation - is an iterative decoding algorithm that uses the structure of the Tanner graph.
- For each iteration of the algorithm:
  - Each bit node sends a message ("extrinsic information") to each check node it's connected to.
  - Each check node sends a message ("extrinsic information") to each bit node it's connected to.
    - "Extrinsic" in this context means we do not pass to a node information the receiving node already possesses.
  - For each codeword bit, we compute the a posteriori probability that the bit takes on the value "1", given all the $Y_i$'s and given that the parity constraints must be met.

# Message Passing, continued

- Let $q_{i,j}(x)$ denote the message passed from bit node $X_i$ to check node $f_j$.
- Then $q_{i,j}(x)$ equals the probability that $X_i = x$ given the channel sample $Y_i$ and all the extrinsic information passed to $X_i$ from all the check nodes *except* $f_j$.

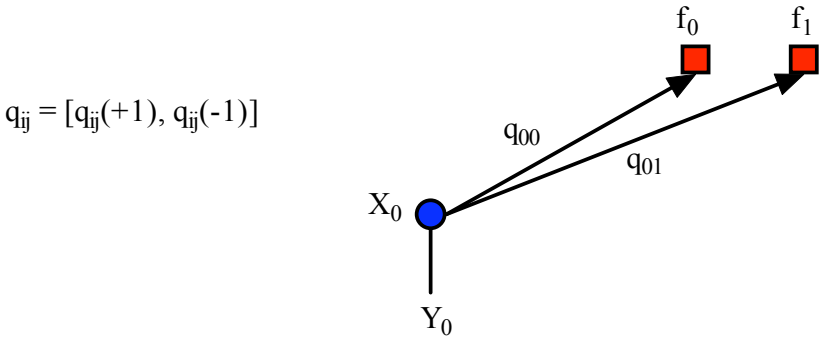



Figure 5: Example of message passed from a bit node to a check node.

# Message Passing, continued

- Let $r_{j,i}(x)$ denote the message passed from check node $f_j$ to bit node $X_i$.
- Then $r_{j,i}(x)$ equals the probability that parity check $f_j$ is satisfied, given $X_i = x$ and given that the other bits connected to $f_j$ (other than $X_i$) have a distribution indicated by the messages they send to $f_j$.
  - Realize: $r_{j,i}(-1) = 1 - r_{j,i}(+1)$

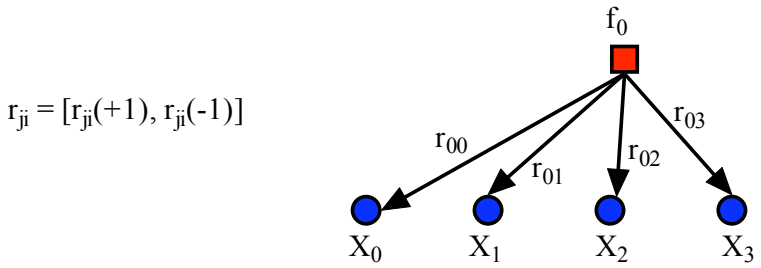
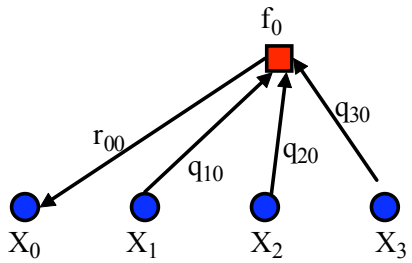
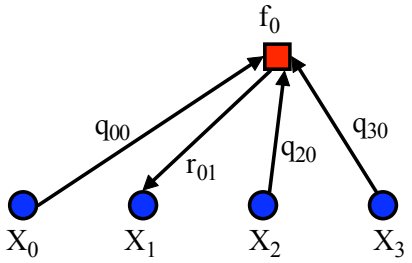


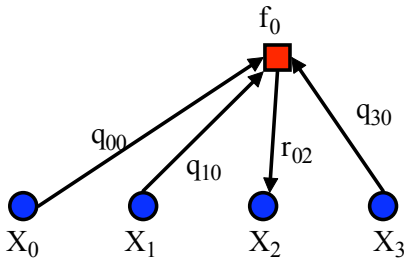

Figure 6: Example of message passed from a bit node to a check node.

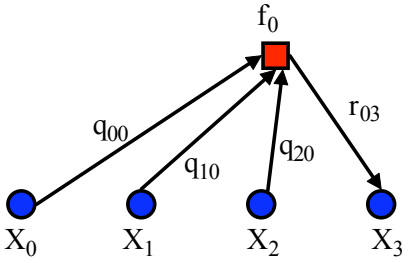
# Message Passing, continued

- So the dependencies of information flow into and out of a check node look like this:



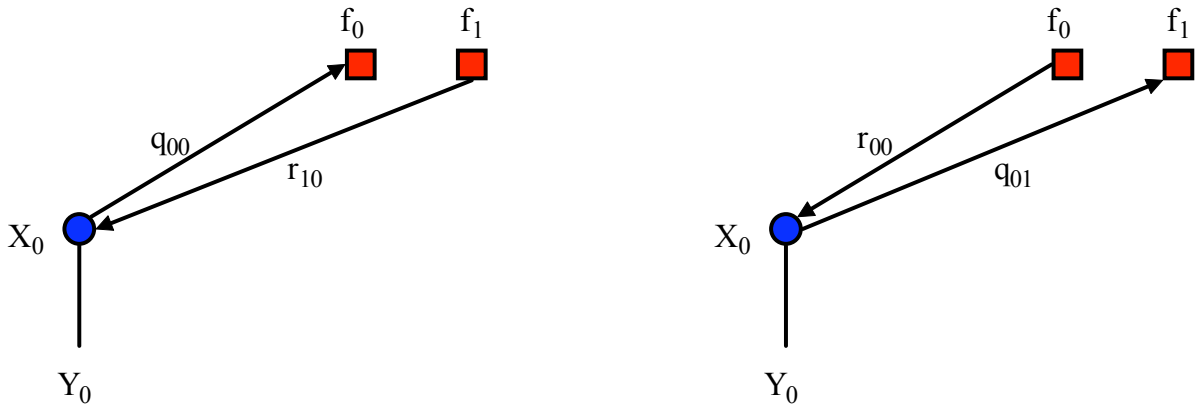



# Message Passing, continued

- And the dependencies of information flow into and out of a bit node look like this:



# Message Passing, continued

**So:**

- How do we initialize the algorithm?
- How do we update
  - The $q_{i,j}$'s from the $r_{j,i}$'s?
  - The $r_{j,i}$'s from the $q_{i,j}$'s?
- How do we compute our estimate of the *a posteriori* probabilities on the $X_i$'s from the $q_{i,j}$'s and the $r_{j,i}$'s.
- How do we stop the algorithm?

# Message Passing, continued

## Algorithm Initialization:

- For $i = 0, 1, \ldots, n-1$, set $q_{i,j}(x)$ based on the observed $Y_i = y_i$ assuming the $X_i$'s are *a priori* equally likely to be $+1$ or $-1$:

$$\begin{aligned} q_{i,j}(+1) &= P(X_i = +1 | Y_i = y_i) \\ &= \frac{f_{Y_i}(y_i | X_i = +1) P(X_i = +1)}{f_{Y_i}(y_i)} \\ &= \frac{(1/\sqrt{2\sigma^2}) \exp[-(y_i - 1)^2/2\sigma^2](1/2)}{(1/2)(1/\sqrt{2\sigma^2})\{\exp[-(y_i - 1)^2/2\sigma^2] + \exp[-(y_i + 1)^2/2\sigma^2]\}} \\ &= \frac{1}{1 + \exp[-2y_i/\sigma^2]}. \end{aligned}$$

# Message Passing, continued

- Similarly, we initialize

$$q_{i,j}(-1) = P(X_i = -1|Y_i = y_i) = \frac{1}{1 + \exp[2y_i/\sigma^2]}.$$

- **Bottom Line:** We initialize the values of $q_{i,j}(x)$ for all $i = 0, 1, \ldots, n-1$ and all $j$ such that there is an edge between bit node $i$ and check node $j$ as follows:

$$q_{i,j}(x) = \frac{1}{1 + \exp[-2xy_i/\sigma^2]} \qquad \text{for } x \in \{+1, -1\}.$$

# Message Passing, continued

**Next Question:** How to iterate to compute APP's?

**The Set-Up:**

- The codeword $\underline{X} = [X_0, X_1, \ldots, X_{n-1}]$ is transmitted. $(X_i \in \{+1, -1\})$
- We observe the received values $[Y_0, Y_1, \ldots, Y_{n-1}] = [y_0, y_1, \ldots, y_{n-1}]$ where $Y_i = X_i + Z_i$, and $\{Z_i\}$ is a sequence of i.i.d. Gaussian random variables with mean zero and variance $\sigma^2$.
- $p_i = P(X_i = -1 | Y_i = y_i) = 1/(1 + \exp[2y_i/\sigma^2])$
- Let $R_j$ denote the location of the 1's in row $j$ of $H$.
- Let $C_i$ denote the location of the 1's in column $i$ of $H$.

# Message Passing, continued

- $R_{j\backslash i} = R_j \backslash \{i\}$
- $C_{i\backslash j} = C_i \backslash \{j\}$
- $X_{k,j}(i) = k^{\text{th}}$ bit in the $j^{\text{th}}$ parity check involving code bit $X_i$. (So $j \in C_i$ and $k \in R_j$).
- $Y_{k,j}(i)$ is a noisy version of $X_{k,j}(i)$.
- $p_{k,j}(i) = P(X_{k,j}(i) = -1 | Y_{k,j}(i) = y_{k,j}(i))$.

# Message Passing, continued

**A Preliminary Lemma:** (Gallager) Consider a sequence of $L$ independent binary-valued random variables $\underline{A} = [A_1, A_2, \ldots, A_L]$ where $P(A_i = 1) = p_i$. Then

$$P(\underline{A} \text{ has even parity}) = \frac{1}{2} + \frac{1}{2}\prod_{i=1}^{L}(1 - 2p_i)$$

and

$$P(\underline{A} \text{ has odd parity}) = \frac{1}{2} - \frac{1}{2}\prod_{i=1}^{L}(1 - 2p_i).$$

Proof: Induction on $L$.

**Note:** With bipolar notation (i.e., $0 \rightarrow +1$ and $1 \rightarrow -1$), "even parity" means the product of the variables is $+1$ and "odd parity" means the product of the variables is $-1$.

# Message Passing, continued

**Theorem:** The *a posteriori* probability (APP) likelihood ratio for $X_i$ given the received word $\mathbf{y} = [y_0, y_1, \ldots, y_{n-1}]$ and given the event $S_i = \{$the bits in $\underline{X}$ satisfy the parity check constraints involving $X_i\}$, is given by

$$\frac{P(X_i = +1|\mathbf{y}, S_i)}{P(X_i = -1|\mathbf{y}, S_i)} = \frac{(1-p_i)}{p_i} \frac{\prod_{j \in C_i} \left(1 + \prod_{i' \in R_{j \backslash i}} (1 - 2p_{i'j}(i))\right)}{\prod_{j \in C_i} \left(1 - \prod_{i' \in R_{j \backslash i}} (1 - 2p_{i'j}(i))\right)}.$$

**Proof:** From Bayes rule:

$$\frac{P(X_i = +1|\mathbf{y}, S_i)}{P(X_i = -1|\mathbf{y}, S_i)} = \frac{\overbrace{P(X_i = +1|y_i)}^{1-p_i} P(S_i|X_i = +1, \mathbf{y})}{\underbrace{P(X_i = -1|y_i)}_{p_i} P(S_i|X_i = -1, \mathbf{y})}.$$

# Message Passing, continued

Consider the term $P(S_i|X_i = +1, \mathbf{y})$:

- Given $X_i = +1$, $S_i$ holds if each of $w_c$ parity checks involving $X_i$ has this property: The $w_r - 1$ bits in the check *other than* $X_i$ have even parity.
- For parity check $j \in C_i$, the probability that the $w_r - 1$ bits other than $X_i$ have even parity is given by the lemma to be:

$$\frac{1}{2} + \frac{1}{2} \prod_{i' \in R_{j \backslash i}} (1 - 2p_{i'j}(i)).$$

# Message Passing, continued

- The independence of the $Y_i$'s means that the probability that *all* $w_c$ parity checks involving $X_i$ are satisfied (given $X_i = +1$) is just

$$P(S_i|X_i = +1, \mathbf{y}) = \prod_{j \in C_i} \left( \frac{1}{2} + \frac{1}{2} \prod_{i' \in R_{j \setminus i}} (1 - 2p_{i'j}(i)) \right).$$

- Similar analysis assuming $X_i = -1$ yields

$$P(S_i|X_i = -1, \mathbf{y}) = \prod_{j \in C_i} \left( \frac{1}{2} - \frac{1}{2} \prod_{i' \in R_{j \setminus i}} (1 - 2p_{i'j}(i)) \right).$$

**QED**

# Message Passing, continued

## Getting Back to Message Passing:

- Recall that $r_{j,i}(+1)$ is the probability that parity check $j$ is satisfied, given that $X_i = +1$ and the other bits in check $j$ have distributions given by $q$. Then from Gallager's lemma:

$$r_{j,i}(+1) = \frac{1}{2} + \frac{1}{2} \prod_{i' \in R_{j \setminus i}} (1 - 2q_{i',j}(-1))$$

  and $r_{j,i}(-1) = 1 - r_{j,i}(+1)$.

- And $q_{i,j}(+1)$ is the probability $X_i = +1$ given $Y_i = y_i$ and the information from the check nodes *other than* the $j^{\text{th}}$ check node - so, as in the theorem,

$$\frac{q_{i,j}(+1)}{q_{i,j}(-1)} = \frac{(1-p_i)}{p_i} \frac{\prod_{j' \in C_{i \setminus j}} r_{j',i}(+1)}{\prod_{j' \in C_{i \setminus j}} r_{j',i}(-1)}.$$

# Message Passing, continued

## Message Passing in the Probability Domain

1. Initialize: For all $i$ and $j$ such that bit $X_i$ is included in parity check $f_j$ - i.e., $h_{j,i} = 1$:
   - Set $p_i = P(X_i = -1|Y_i = y_i) = 1/(1 + \exp(2y_i/\sigma^2))$.
   - $q_{i,j}(+1) = 1 - p_i$.
   - $q_{i,j}(-1) = p_i$.
2. Pass information from check nodes to bit nodes:
   - $r_{j,i}(+1) = \frac{1}{2} + \frac{1}{2}\prod_{i' \in R_{j \setminus i}}(1 - 2q_{i',j}(-1))$
   - $r_{j,i}(-1) = 1 - r_{j,i}(+1)$.

# Message Passing, continued

3. Pass information from bit nodes to check nodes:
   - $q_{i,j}(+1) = K_{i,j}(1-p_i)\prod_{j' \in C_{i\backslash j}} r_{j',i}(+1)$
   - $q_{i,j}(-1) = K_{i,j}p_i\prod_{j' \in C_{i\backslash j}} r_{j',i}(-1)$

   Here, the constants $K_{i,j}$ are chosen to guarantee that $q_{i,j}(+1)+q_{i,j}(-1) = 1$.
4. Compute the APP likelihood ratios for each bit position $i$:
   - $Q_i(+1) = K_i(1-p_i)\prod_{j \in C_i} r_{j,i}(+1)$
   - $Q_i(-1) = K_i p_i\prod_{j \in C_i} r_{j,i}(-1)$

   Here, the constants $K_i$ are chosen to guarantee that $Q_i(+1) + Q_i(-1) = 1$.
5. Compute the hard decisions and decide if it's time to stop.

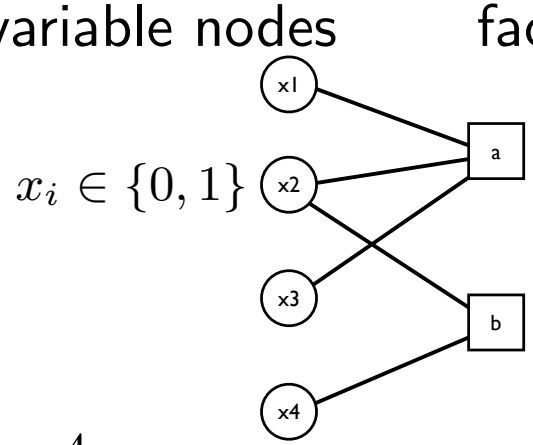
$$\hat{X}_i = \begin{cases} +1, & \text{if } Q_i(+1) \geq 0.5; \\ -1, & \text{otherwise.} \end{cases}$$

# Message Passing, continued

If all parity checks satisfied or the maximum number of iterations reached then stop; otherwise, go to (2).

# Example: decoding LDPC codes

- LDPC code is defined by a factor graph model




  - block length $n = 4$
  - number of factors $m = 2$
  - allowed messages $= \{0000, 0111, 1010, 1101\}$

- decoding using belief propagation (for BSC with $\epsilon = 0.3$)

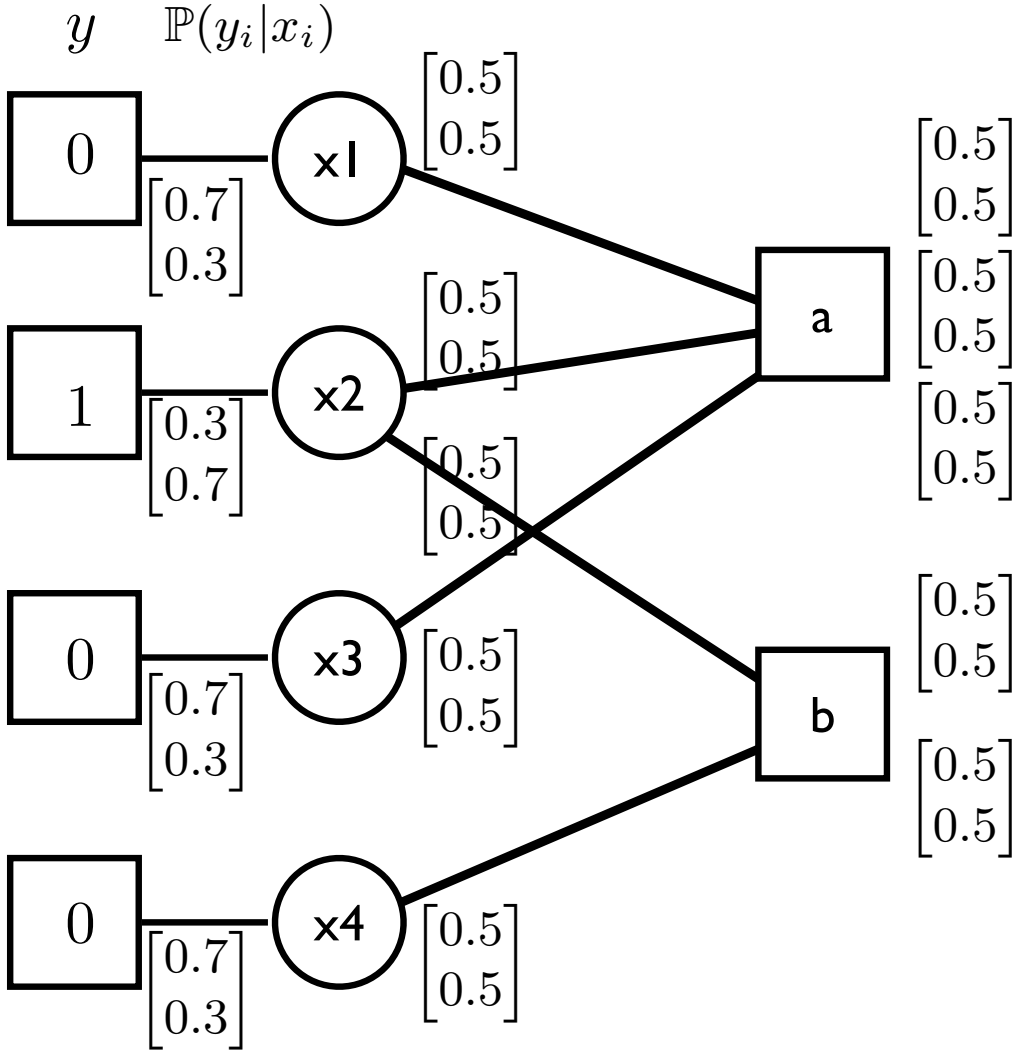
$$\mu_y(x) = \frac{1}{Z} \prod_{i \in V} \mathbb{P}_{Y|X}(y_i|x_i) \prod_{a \in F} \mathbb{I}(\oplus x_{\partial a} = 0)$$

- use (parallel) sum-product algorithm to find $\mu(x_i)$ and let

$$\hat{x}_i = \arg\max \mu(x_i)$$

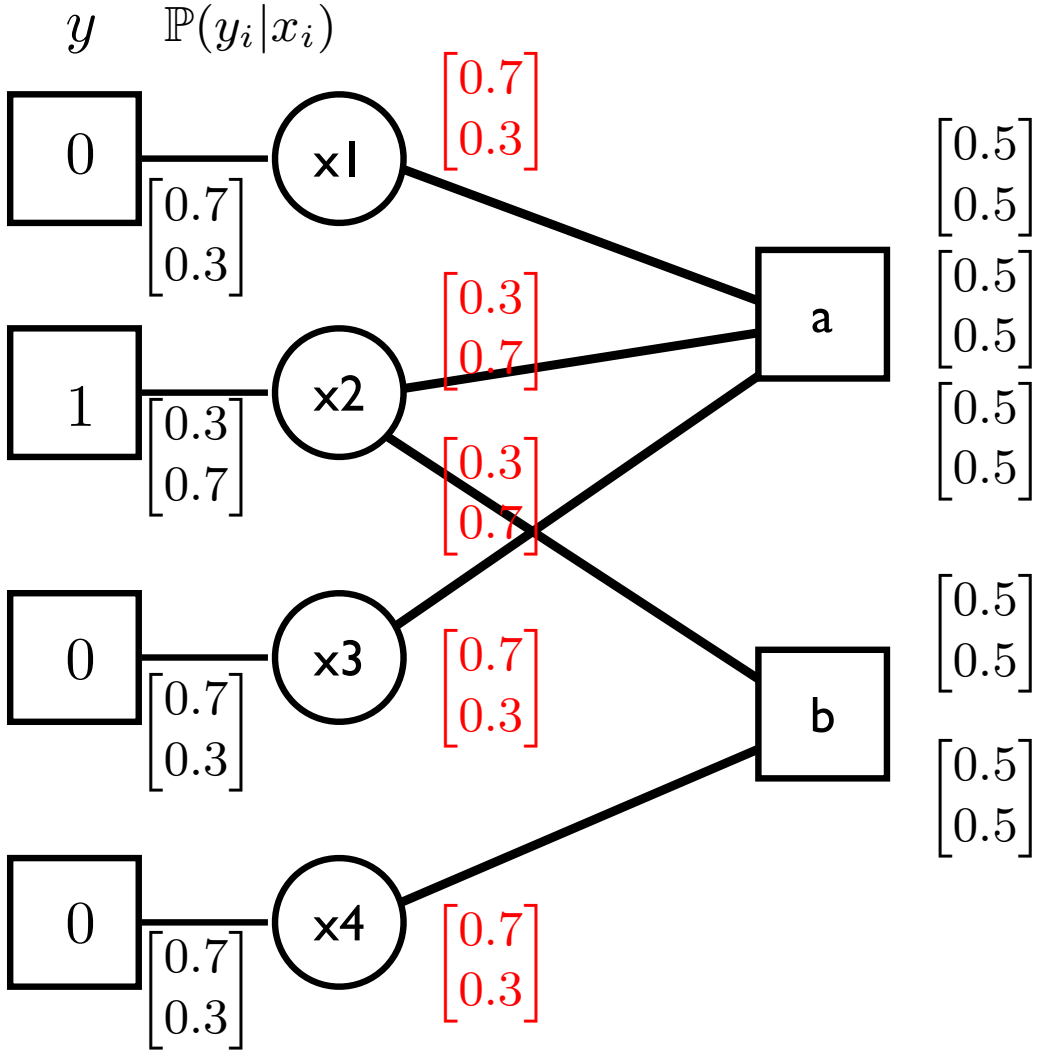
  - minimizes bit error rate

# Decoding by sum-product algorithm



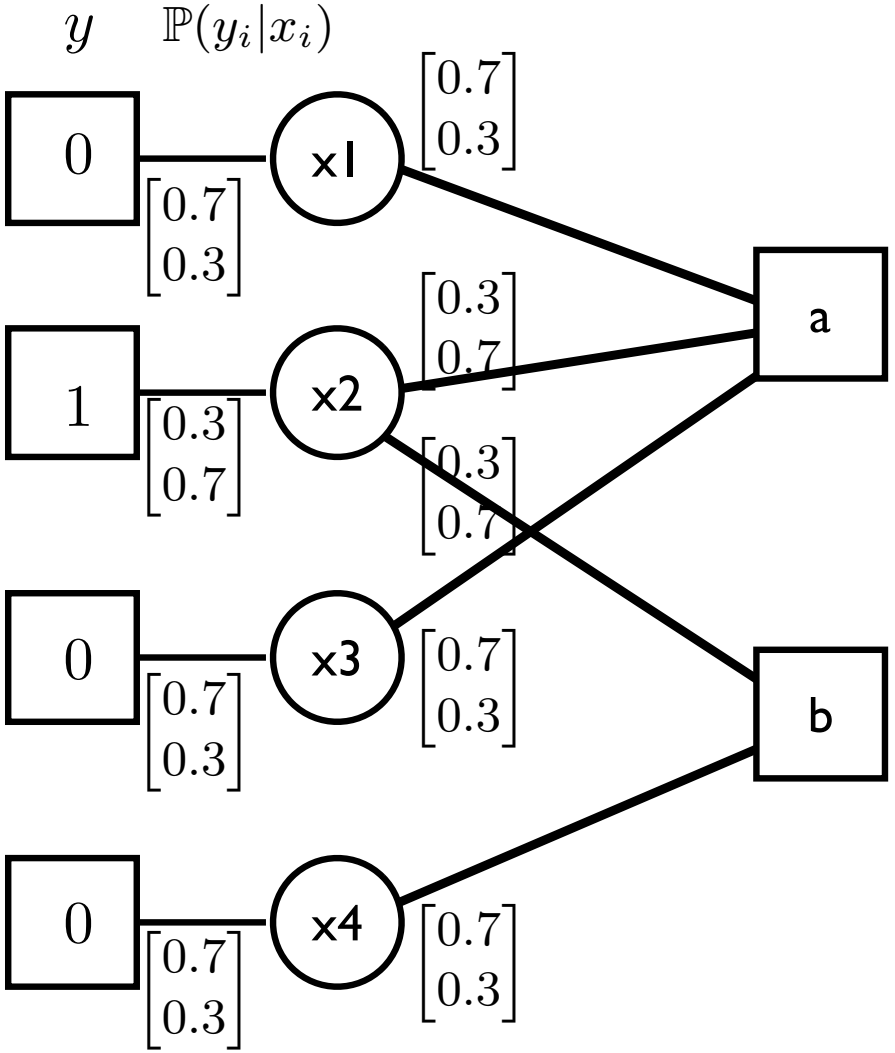

$$\nu_{i\to a}^{(t+1)}(x_i) = \mathbb{P}(y_i|x_i)\prod_{b\in\partial i\setminus\{a\}}\tilde{\nu}_{b\to i}^{(t)}(x_i)$$
$$\tilde{\nu}_{a\to i}^{(t+1)}(x_i) = \sum_{x_{\partial a\setminus\{i\}}}\prod_{j\in\partial a\setminus\{i\}}\nu_{j\to a}(x_j)\mathbb{I}(\oplus x_{\partial a}=0)$$

# Decoding by sum-product algorithm



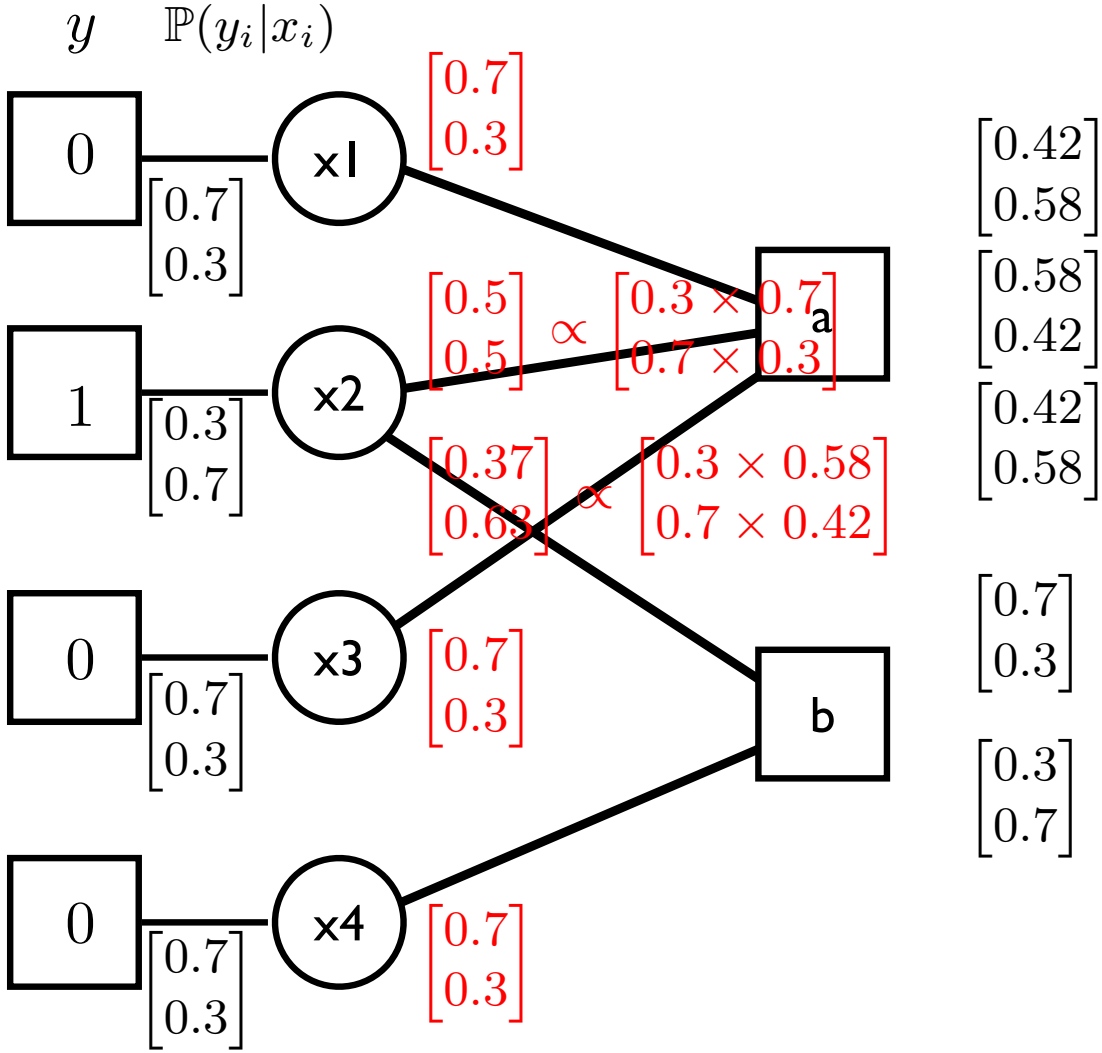

$$\nu_{i\to a}^{(t+1)}(x_i) = \mathbb{P}(y_i|x_i)\prod_{b\in\partial i\setminus\{a\}}\tilde{\nu}_{b\to i}^{(t)}(x_i)$$
$$\tilde{\nu}_{a\to i}^{(t+1)}(x_i) = \sum_{x_{\partial a\setminus\{i\}}}\prod_{j\in\partial a\setminus\{i\}}\nu_{j\to a}(x_j)\mathbb{I}(\oplus x_{\partial a}=0)$$

# Decoding by sum-product algorithm



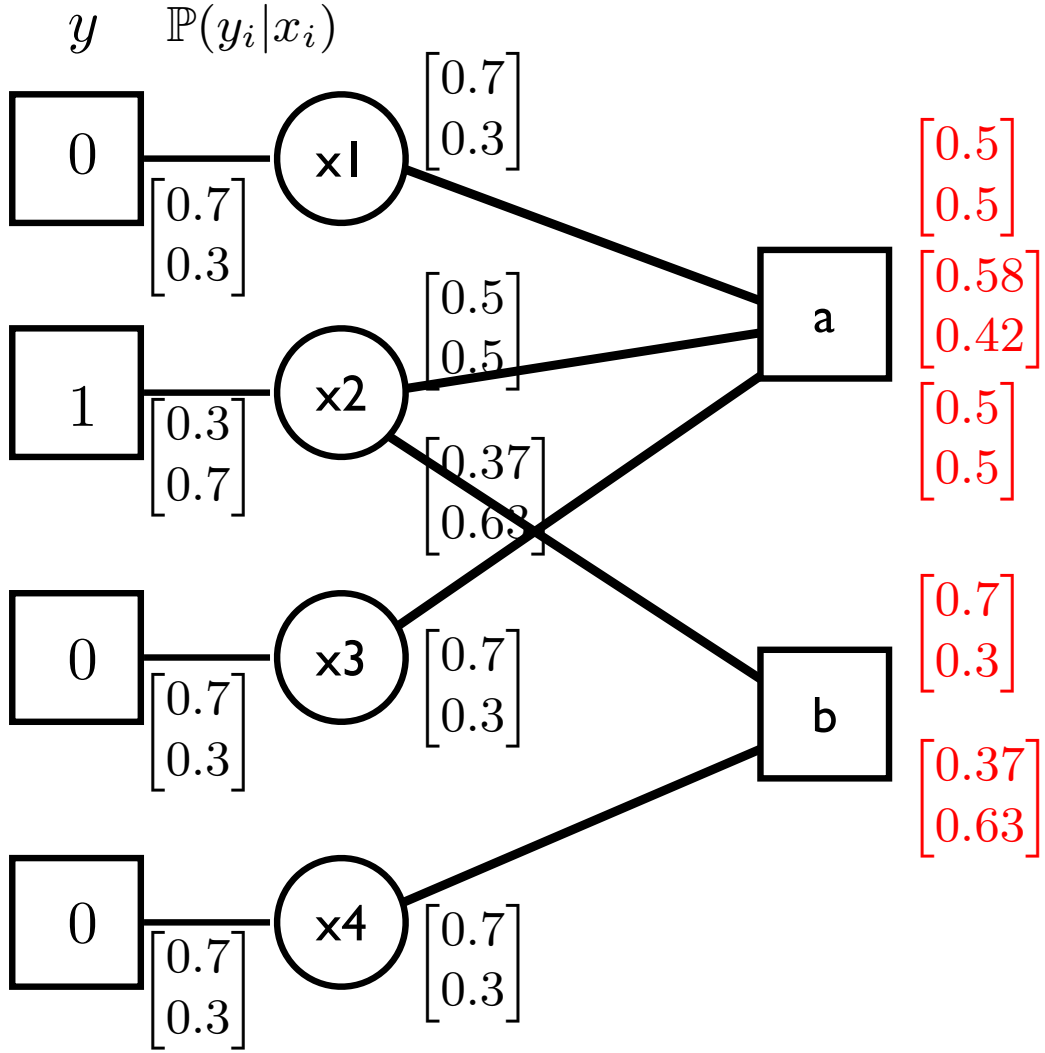

$$\nu_{i\to a}^{(t+1)}(x_i) = \mathbb{P}(y_i|x_i)\prod_{b\in\partial i\setminus\{a\}}\tilde{\nu}_{b\to i}^{(t)}(x_i)$$
$$\tilde{\nu}_{a\to i}^{(t+1)}(x_i) = \sum_{x_{\partial a\setminus\{i\}}}\prod_{j\in\partial a\setminus\{i\}}\nu_{j\to a}(x_j)\mathbb{I}(\oplus x_{\partial a}=0)$$

# Decoding by sum-product algorithm



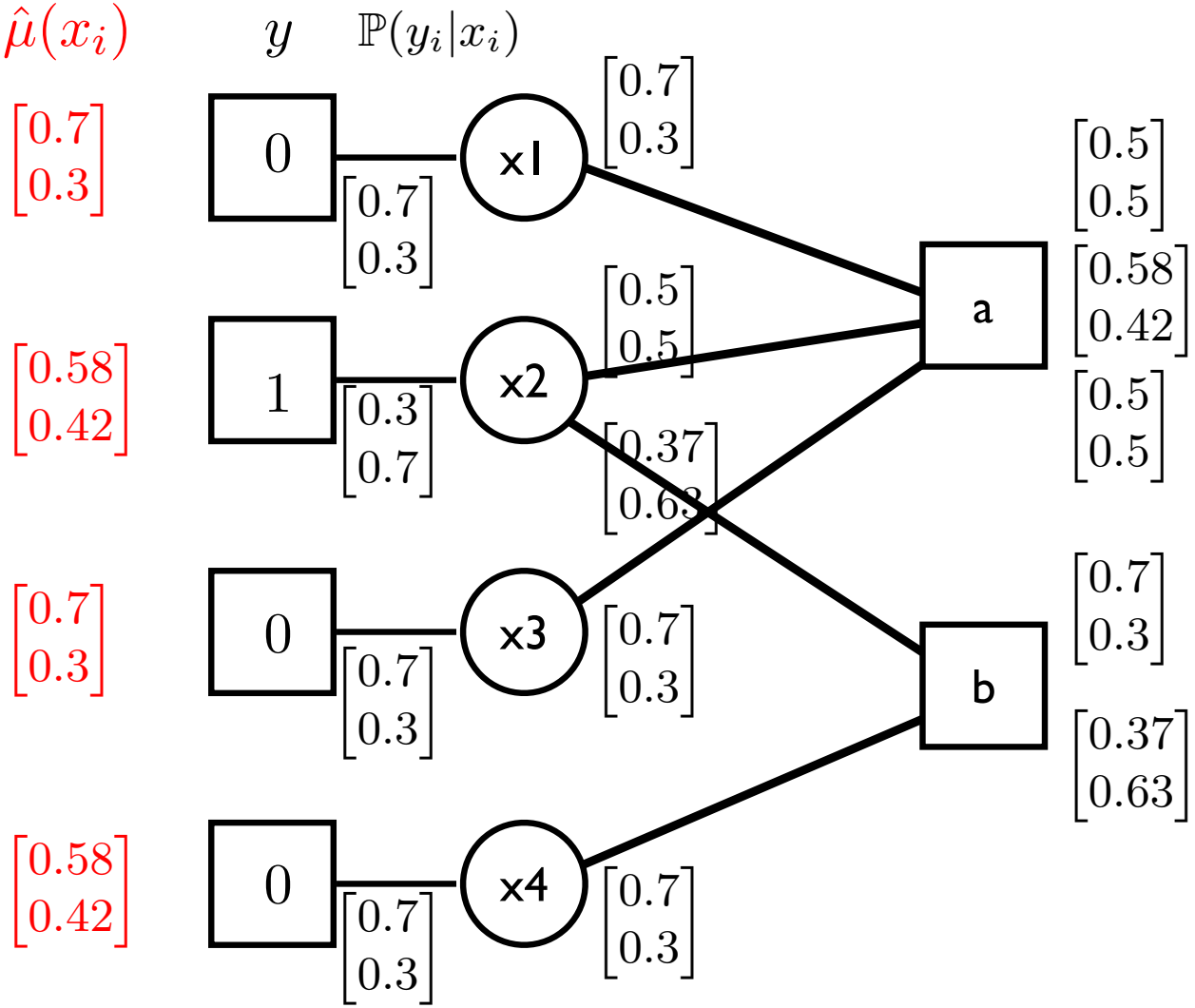

$$\nu_{i\to a}^{(t+1)}(x_i) = \mathbb{P}(y_i|x_i)\prod_{b\in\partial i\setminus\{a\}}\tilde{\nu}_{b\to i}^{(t)}(x_i)$$
$$\tilde{\nu}_{a\to i}^{(t+1)}(x_i) = \sum_{x_{\partial a\setminus\{i\}}}\prod_{j\in\partial a\setminus\{i\}}\nu_{j\to a}(x_j)\mathbb{I}(\oplus x_{\partial a}=0)$$

# Decoding by sum-product algorithm




$$\nu_{i\to a}^{(t+1)}(x_i) = \mathbb{P}(y_i|x_i)\prod_{b\in\partial i\setminus\{a\}}\tilde{\nu}_{b\to i}^{(t)}(x_i)$$
$$\tilde{\nu}_{a\to i}^{(t+1)}(x_i) = \sum_{x_{\partial a\setminus\{i\}}}\prod_{j\in\partial a\setminus\{i\}}\nu_{j\to a}(x_j)\mathbb{I}(\oplus x_{\partial a}=0)$$

# Decoding by sum-product algorithm




$$\nu_{i\to a}^{(t+1)}(x_i) = \mathbb{P}(y_i|x_i)\prod_{b\in\partial i\setminus\{a\}}\tilde{\nu}_{b\to i}^{(t)}(x_i)$$
$$\tilde{\nu}_{a\to i}^{(t+1)}(x_i) = \sum_{x_{\partial a\setminus\{i\}}}\prod_{j\in\partial a\setminus\{i\}}\nu_{j\to a}(x_j)\mathbb{I}(\oplus x_{\partial a}=0)$$

# Message Passing, continued

**An Example:** Consider the "sum code" shown below:

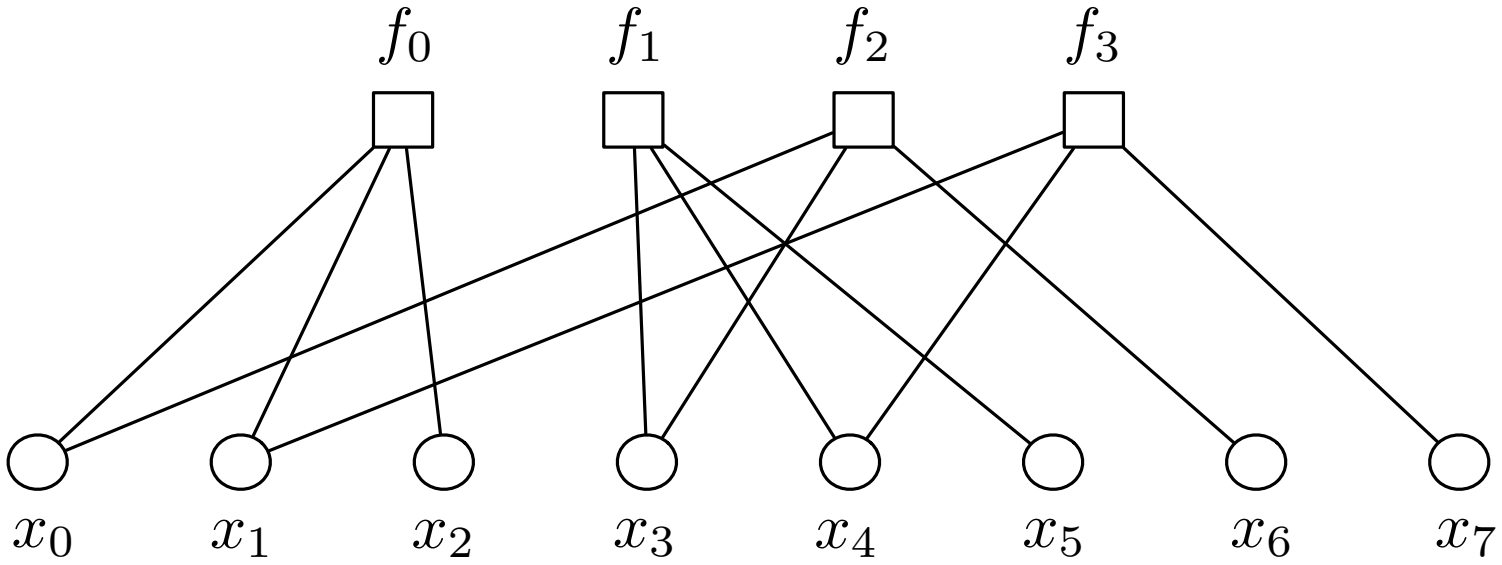
| | | |
|---|---|---|
| $c_0$ | $c_1$ | $c_2$ |
| $c_3$ | $c_4$ | $c_5$ |
| $c_6$ | $c_7$ | |

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

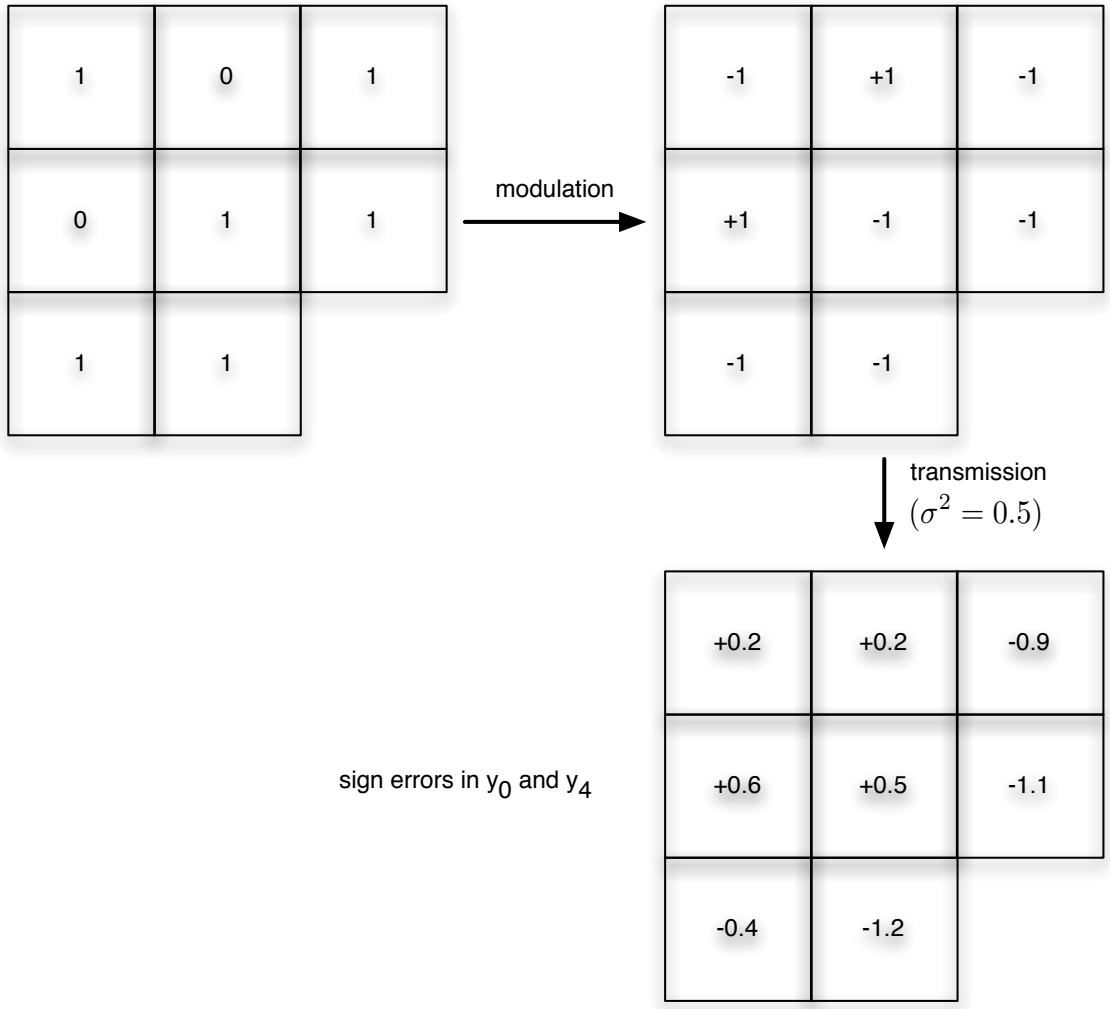
- $n = 8$, $m = n - k = 4$, and $d_{\min} = 3$.

# Message Passing, continued

- Neither low-density nor regular.



# Message Passing, continued



# Message Passing, continued

Initialization: $q_{i,j}(x) = 1/(1+\exp(-2xy_i/\sigma^2))$ for each $i, j$ such that $h_{j,i} = 1$.

- $q_{0,0}(-1) = q_{0,2}(-1) = 0.310$ and $q_{0,0}(+1) = q_{0,2}(+1) = 0.690$.
- $q_{1,0}(-1) = q_{1,3}(-1) = 0.310$ and $q_{1,0}(+1) = q_{1,3}(+1) = 0.690$.
- $q_{2,0}(-1) = 0.973$ and $q_{2,0}(+1) = 0.027$.
- $q_{3,1}(-1) = q_{3,2}(-1) = 0.083$ and $q_{3,1}(+1) = q_{3,2}(+1) = 0.917$.
- $q_{4,1}(-1) = q_{4,3}(-1) = 0.119$ and $q_{4,1}(+1) = q_{4,3}(+1) = 0.881$.
- $q_{5,1}(-1) = 0.988$ and $q_{5,1}(+1) = 0.012$.
- $q_{6,2}(-1) = 0.832$ and $q_{6,2}(+1) = 0.168$.
- $q_{7,3}(-1) = 0.992$ and $q_{7,3}(+1) = 0.008$.

# Message Passing, continued

Now compute $r_{j,i}$'s from $q_{i,j}$'s:

$$\begin{aligned} r_{0,0}(+1) &= \frac{1}{2} + \frac{1}{2} \prod_{i' \in R_{0\backslash 0}} (1 - 2q_{i',0}(-1)) \\ &= \frac{1}{2} + \frac{1}{2}(1 - 2q_{1,0}(-1))(1 - 2q_{2,0}(-1)) \\ &= \frac{1}{2} + \frac{1}{2}(1 - 2(0.31))(1 - 2(0.973)) \\ &= 0.320. \end{aligned}$$

# Message Passing, continued

In a similar way:

- $r_{0,1}(+1) = 0.5 + 0.5(1 - 2(0.31))(1 - 2(0.973)) = 0.32$
- $r_{0,2}(+1) = 0.5 + 0.5(1 - 2(0.31))(1 - 2(0.31)) = 0.57$
- $r_{1,3}(+1) = 0.5 + 0.5(1 - 2(0.119))(1 - 2(0.988)) = 0.128$
- $r_{2,0}(+1) = 0.5 + 0.5(1 - 2(0.083))(1 - 2(0.832)) = 0.223$
- etc.

And, of course, $r_{j,i}(-1) = 1 - r_{j,i}(+1)$.

# Message Passing, continued

Now compute $q_{i,j}$'s from $r_{j,i}$'s:

$$\begin{aligned}\tilde{q}_{0,0}(+1) &= (1-p_0)\prod_{j'\in C_{0\backslash 0}} r_{j',0}(+1)\\ &= (0.69)r_{2,0}(+1)\\ &= (0.69)(0.223) = 0.154\end{aligned}$$

and

$$\begin{aligned}\tilde{q}_{0,0}(-1) &= p_0\prod_{j'\in C_{0\backslash 0}} r_{j',0}(-1)\\ &= (0.31)r_{2,0}(-1)\\ &= (0.31)(0.777) = 0.241.\end{aligned}$$

# Message Passing, continued

This means

$$q_{0,0}(+1) = \frac{0.154}{0.154 + 0.241} = 0.39 \text{ and } q_{0,0}(-1) = \frac{0.241}{0.154 + 0.241} = 0.61.$$

Finally, compute the APP's:

- Note: $\tilde{Q}_i(+1) = \tilde{q}_{i,j}(+1) r_{j,i}(+1)$, which means

$$\tilde{Q}_0(+1) = \tilde{q}_{0,0}(+1) r_{0,0}(+1) = 0.154 \times 0.32 = 0.0493$$

and

$$\tilde{Q}_0(-1) = \tilde{q}_{0,0}(-1) r_{0,0}(-1) = 0.241 \times 0.68 = 0.164.$$

# Message Passing, continued

This yields the APP

$$Q_0(+1) = \frac{0.0493}{0.0493 + 0.164} = 0.23$$

and

$$Q_0(-1) = \frac{0.164}{0.0493 + 0.164} = 0.77.$$

- The other $Q_i$'s can be computed similarly.

# Message Passing, continued

- Results $[Q \triangleq (Q_0(-1), \ldots, Q_7(-1))]$

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| iteration=1 | | | | | | | | |
| $Q =$ | 0.7686 | 0.8694 | 0.9647 | 0.5076 | 0.7426 | 0.9479 | 0.7199 | 0.9853 |
| $c =$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| iteration=2 | | | | | | | | |
| $Q =$ | 0.1751 | 0.1836 | 0.9668 | 0.2330 | 0.4637 | 0.9722 | 0.8305 | 0.9919 |
| $c =$ | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| iteration=3 | | | | | | | | |
| $Q =$ | 0.7232 | 0.4609 | 0.9667 | 0.6308 | 0.8091 | 0.9498 | 0.6802 | 0.9909 |
| $c =$ | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| iteration=4 | | | | | | | | |
| $Q =$ | 0.5307 | 0.2683 | 0.9731 | 0.1477 | 0.4095 | 0.9811 | 0.8392 | 0.9922 |
| $c =$ | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| iteration=5 | | | | | | | | |
| $Q =$ | 0.7564 | 0.5799 | 0.9672 | 0.3425 | 0.7573 | 0.9558 | 0.8102 | 0.9896 |
| $c =$ | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| iteration=6 | | | | | | | | |
| $Q =$ | 0.4544 | 0.2089 | 0.9736 | 0.2136 | 0.6243 | 0.9686 | 0.8412 | 0.9924 |
| $c =$ | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| iteration=7 | | | | | | | | |
| $Q =$ | 0.7399 | 0.3381 | 0.9692 | 0.4086 | 0.7869 | 0.9567 | 0.7754 | 0.9923 |
| $c =$ | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |

- Note: converges to the correct codeword after 7 iterations.

# Message Passing, continued

**Fact:** As with the Viterbi and BCJR algorithms, message passing is often carried out using not probabilities but rather the logarithms of (ratios of) probabilities.

**Why?**

- Costly multiplications become less costly additions
- Numerical stability is easier to control.

**So:** Define the following quantities:

$$L(X_i) = \log \frac{P(X_i = +1|Y_i = y_i)}{P(X_i = -1|Y_i = y_i)} \qquad L(r_{j,i}) = \log \frac{r_{j,i}(+1)}{r_{j,i}(-1)}$$

$$L(q_{i,j}) = \log \frac{q_{i,j}(+1)}{q_{i,j}(-1)} \qquad L(Q_i) = \log \frac{Q_i(+1)}{Q_i(-1)}$$

**Our Task:** Reformulate message passing in terms of these quantities.

# Message Passing, continued

**Step 1: Initialization:** Assume $X_i$ is *a priori* equally likely to be $+1$ or $-1$ and compute the log-APP based on the observed value of $Y_i$:

$$\begin{aligned} L(q_{i,j}) &= \log \frac{P(X_i = +1|Y_i = y_i)}{P(X_i = -1|Y_i = y_i)} \\ &= \log \frac{1/(1+\exp(-2y_i/\sigma^2))}{1/(1+\exp(2y_i/\sigma^2))} \\ &= \log \frac{1+\exp(2y_i/\sigma^2)}{1+\exp(-2y_i/\sigma^2)} \\ &= \log \exp(2y_i/\sigma^2) \frac{1+\exp(-2y_i/\sigma^2)}{1+\exp(-2y_i/\sigma^2)} \\ &= 2y_i/\sigma^2. \end{aligned}$$

# Message Passing, continued

**Step 2: Computation of messages sent from check nodes to bit nodes:**

- Recall that $r_{j,i}(-1) = \frac{1}{2} - \frac{1}{2}\prod_{i' \in R_{j\backslash i}}(1 - 2q_{i'j}(-1))$ which implies that

$$1 - 2r_{j,i}(-1) = \prod_{i' \in R_{j\backslash i}} (1 - 2q_{i'j}(-1)).$$

- A useful identity: If $x$ and $y$ are positive numbers summing to one, then

$$\tanh\left(\frac{1}{2}\log(x/y)\right) = 1 - 2y.$$

- Applying this identity - first to $x = r_{j,i}(+1)$ and $y = r_{j,i}(-1)$ and then to $x = q_{i'j}(+1)$ and $y = q_{i'j}(-1)$, we get

# Message Passing, continued

$$\tanh\left(\frac{1}{2}L(r_{j,i})\right) = 1 - 2r_{j,i}(-1) = \prod_{i' \in R_{j\backslash i}} (1 - 2q_{i'j}(-1))$$

$$= \prod_{i' \in R_{j\backslash i}} \tanh\left(\frac{1}{2}L(q_{i',j})\right)$$

or

$$L(r_{j,i}) = 2\tanh^{-1}\left(\prod_{i' \in R_{j\backslash i}} \tanh(\frac{1}{2}L(q_{i',j}))\right).$$

- **Problem:** We've still got products (not additions) - *and* we've added hyperbolic functions!

# Message Passing, continued

- Solution: Let $L(q_{i,j}) = \alpha_{i,j}\beta_{i,j}$, where $\alpha_{i,j} = \text{sgn}(L(q_{i,j}))$ and $\beta_{i,j} = |L(q_{i,j})|$:

$$\tanh(\frac{1}{2}L(r_{j,i})) = \prod_{i' \in R_{j\backslash i}} \alpha_{i',j} \prod_{i' \in R_{j\backslash i}} \tanh\left(\frac{1}{2}\beta_{i',j}\right).$$

- So:

$$\begin{aligned} L(r_{i,j}) &= \Big(\prod_{i' \in R_{j\backslash i}} \alpha_{i',j}\Big) \cdot 2\tanh^{-1} \prod_{i' \in R_{j\backslash i}} \tanh(\beta_{i',j}/2) \\ &= \Big(\prod_{i' \in R_{j\backslash i}} \alpha_{i',j}\Big) \cdot \phi\Big(\sum_{i' \in R_{j\backslash i}} \phi(\beta_{i',j})\Big) \end{aligned}$$

where $\phi(x) = -\ln\tanh(x/2) = \ln[(e^{x/2}+e^{-x/2})/(e^{x/2}-e^{-x/2})] = \ln[(e^x+1)/(e^x-1)]$ and we've used the fact that $\phi(\cdot)$ is its own inverse – i.e., $\phi(\phi(x)) = x$.

**Note:** $\tanh(x)$ and $\tanh^{-1}(x)$ are odd functions.

# Message Passing, continued

- So the only product left is a product of $\pm 1$'s - i.e., a parity check.
- Of course, we have to implement the (somewhat ugly) function $\phi(x)$ - just like we had to implement the (somewhat ugly) function $\ln(1+e^{-|x-y|})$ when decoding turbo codes. (There are similar approximations we can make here.)
- **Bottom Line:** we now have a tractable way to compute the $L(r_{j,i})$'s from the $L(q_{i,j})$'s - i.e., a way to compute the messages at the check nodes based on input from the bit nodes.
- What about the other messages?

# Message Passing, continued

- Computing $L(q_{i,j})$:
  - Much easier, because $q_{i,j}$'s are just products of $r_{i,j}$'s.
  - So we can just divide the formula for $q_{i,j}(+1)$ by the formula for $q_{i,j}(-1)$ and take the logarithm:

$$L(q_{i,j}) = L(X_i) + \sum_{j' \in C_{i \setminus j}} L(r_{j',i}).$$

- Computing $L(Q_i)$: Once again, trivial because of the product form of the $Q_i$'s:

$$L(Q_i) = L(X_i) + \sum_{j \in C_i} L(r_{j,i}).$$

# Message Passing, continued

## Message Passing in the Log Domain

1. Initialize: For all $i$ and $j$ such that bit $X_i$ is included in parity check $f_j$ - i.e., for all $(i, j)$ such that $h_{j,i} = 1$:

$$L(q_{i,j}) = L(X_i) = 2y_i/\sigma^2.$$

2. Pass information from check nodes to bit nodes:

$$L(r_{j,i}) = \Big(\prod_{i' \in R_{j\backslash i}} \alpha_{i',j}\Big) \cdot \phi\Big(\sum_{i' \in R_{j\backslash i}} \phi(\beta_{i',j})\Big),$$

where

$$\alpha_{i,j} = \mathrm{sgn}(L(q_{i,j})) \quad \text{and} \quad \beta_{i,j} = |L(q_{i,j})| \quad \text{and} \quad \phi(x) = \log\left(\frac{e^x + 1}{e^x - 1}\right).$$

# Message Passing, continued

3. Pass information from bit nodes to check nodes:

$$L(q_{i,j}) = L(X_i) + \sum_{j' \in C_{i \setminus j}} L(r_{j',i}).$$

4. Compute the log-APP ratios for each bit position $i$:

$$L(Q_i) = L(X_i) + \sum_{j \in C_i} L(r_{j,i}).$$

5. Compute the hard decisions and decide if it's time to stop.

$$\hat{X}_i = \begin{cases} +1, & \text{if } L(Q_i) > 0; \\ -1, & \text{otherwise.} \end{cases}$$

If all parity checks satisfied or the maximum number of iterations reached then stop; otherwise, go to (2).

# Message Passing, continued

- If we repeat the previous example in the log domain, we obtain $[LQ \triangleq (L(Q_0), \ldots, L(Q_7))]$:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| iteration=1 | | | | | | | | |
| $LQ =$ | $-1.2002$ | $-1.8953$ | $-3.3092$ | $-0.0306$ | $-1.0597$ | $-2.9008$ | $-0.9439$ | $-4.2042$ |
| $c =$ | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| iteration=2 | | | | | | | | |
| $LQ =$ | 1.5499 | 1.4922 | $-3.3721$ | 1.1913 | 0.1455 | $-3.5547$ | $-1.5889$ | $-4.8064$ |
| $c =$ | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| iteration=3 | | | | | | | | |
| $LQ =$ | $-0.9605$ | 0.1568 | $-3.3680$ | $-0.5354$ | $-1.4442$ | $-2.9399$ | $-0.7545$ | $-4.6958$ |
| $c =$ | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| iteration=4 | | | | | | | | |
| $LQ =$ | $-0.1229$ | 1.0031 | $-3.5876$ | 1.7531 | 0.3659 | $-3.9473$ | $-1.6520$ | $-4.8420$ |
| $c =$ | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| iteration=5 | | | | | | | | |
| $LQ =$ | $-1.1331$ | $-0.3222$ | $-3.3854$ | 0.6521 | $-1.1379$ | $-3.0733$ | $-1.4512$ | $-4.5529$ |
| $c =$ | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| iteration=6 | | | | | | | | |
| $LQ =$ | 0.1830 | 1.3318 | $-3.6083$ | 1.3031 | $-0.5077$ | $-3.4307$ | $-1.6673$ | $-4.8708$ |
| $c =$ | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| iteration=7 | | | | | | | | |
| $LQ =$ | $-1.0455$ | 0.6718 | $-3.4495$ | 0.3697 | $-1.3064$ | $-3.0952$ | $-1.2390$ | $-4.8631$ |
| $c =$ | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |

# Message Passing, continued

- Note: In the first iteration, $L(Q_0) = -1.2$, in agreement with the probability domain result of $Q_0(-1) = .7686$:

$$\log\left(\frac{1-.7686}{.7686}\right) = -1.2$$

# Performance of Regular LDPC Codes Under Message Passing

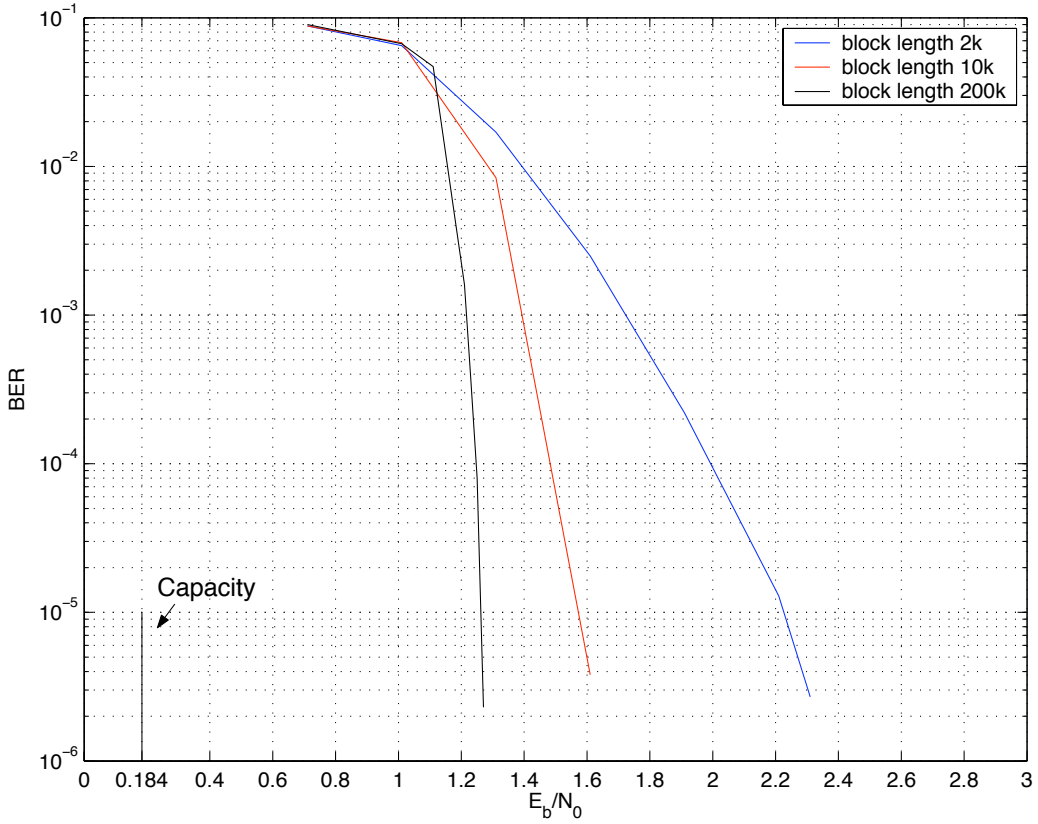



Figure 7: The performance of $(3, 6)$ randomly-designed LDPC codes over a BPSK-modulated AWGN channel for blocklengths of $n = 2000$, 10,000, and 200,000.

# Fundamental Limits to Message Passing

**Observation:** As we increase the blocklength of an $(w_c, w_r)$-regular LDPC code and decode via message-passing, we find that performance over a BPSK-modulated AWGN improves - but that it reaches a limit.

- Not too surprising, since we know that Shannon capacity certainly represents just such a limit.
- But for regular LDPC codes, we can't quite make it to the Shannon limit.

**Fact:** Associated with the values $(w_c, w_r)$ and the BPSK-modulated channel is a number we call the *threshold.*

- The threshold is a measure of channel quality - specified in terms of signal-to-noise ratio for BPSK AWGN.

# Fundamental Limits to Message Passing, continued

- If the channel quality falls below the threshold, then iterative decoding cannot yield arbitrarily good performance; the BER will be bounded away from zero.
- This threshold is bounded by capacity - and for *regular* LDPC codes, it is strictly greater than capacity.
- Moreover, for long blocklengths, we can design LDPC codes whose "waterfall" is close to the threshold.
- A number of other channels exhibiting particular forms of symmetry and other forms of iterative decoding can also be associated with thresholds. (E.g., turbo decoding, binary symmetric channel, particular coded modulation formats, etc.)
- Example: The threshold for $(3, 6)$ regular LDPC codes on the BPSK-modulated AWGN channel is $E_b/N_0 = 1.11$ dB - compared with the Shannon capacity of $E_b/N_0 = 0.184$ dB for BPSK-modulated AWGN.

# Fundamental Limits to Message Passing, continued

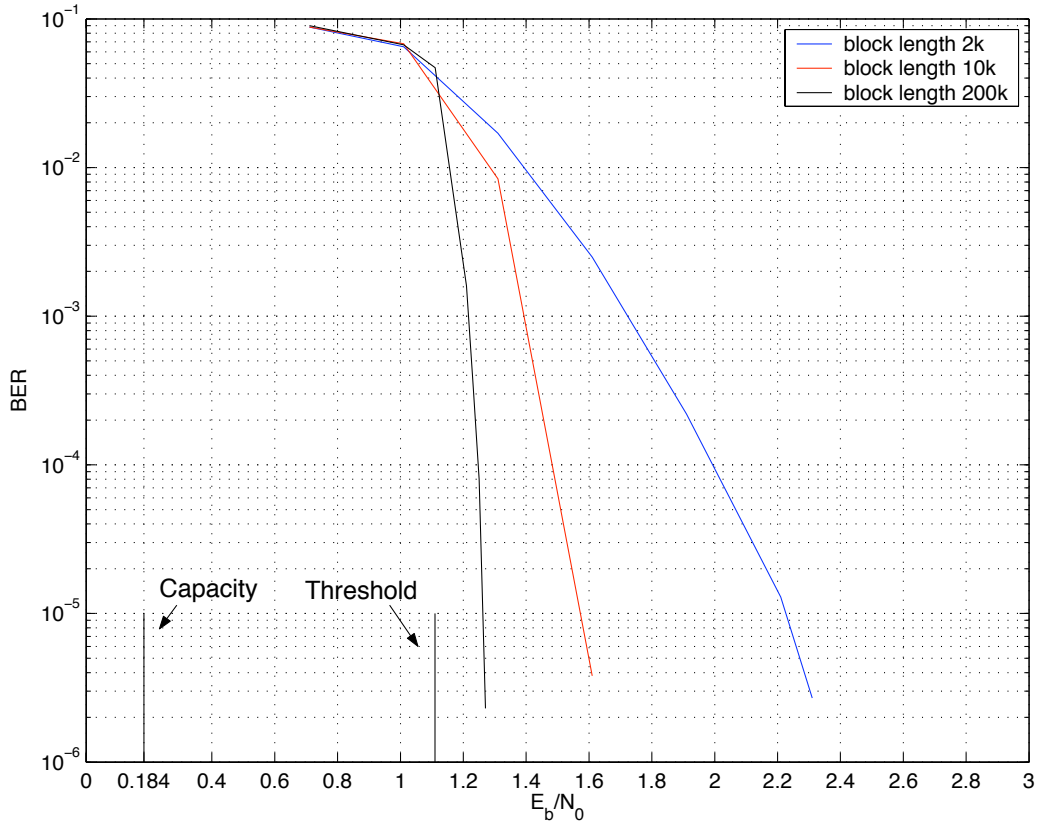



Figure 8: Comparing the performance of $(3, 6)$ LDPC codes with the $(3, 6)$ threshold and the rate-1/2 BPSK AWGN capacity.

# Fundamental Limits to Message Passing, continued

## How Are Thresholds Computed?

- Key insight: Messages are *random variables* with probability density functions (PDF's) that evolve with each iteration.
- So set the channel SNR and then track the PDF's of the messages passed over the Tanner graph - "density evolution".
- The lowest channel SNR at which the messages converge to the correct value is called the threshold.
  - If message passing is done in the log domain, then the messages "converge" when the log-likelihood values "blow up".

# Fundamental Limits to Message Passing, continued

- Important "threshold" papers:
  - Tom Richardson and Rudiger Urbanke, "The Capacity of Low-Density Parity Check Codes Under Message-Passing Decoding," *IEEE Trans. on Info. Theory,* February 2001.
  - H. El Gamal and A. R. Hammons, "Analyzing the Turbo Decoder Using the Gaussian Approximation," *IEEE Trans. on Info. Theory,* February 2001.
  - S. ten Brink, "Convergence Behavior of Iteratively Decoded Parallel Concatenated Codes," *IEEE Trans. on Comm.,* October 2001.

# Fundamental Limits to Message Passing, continued

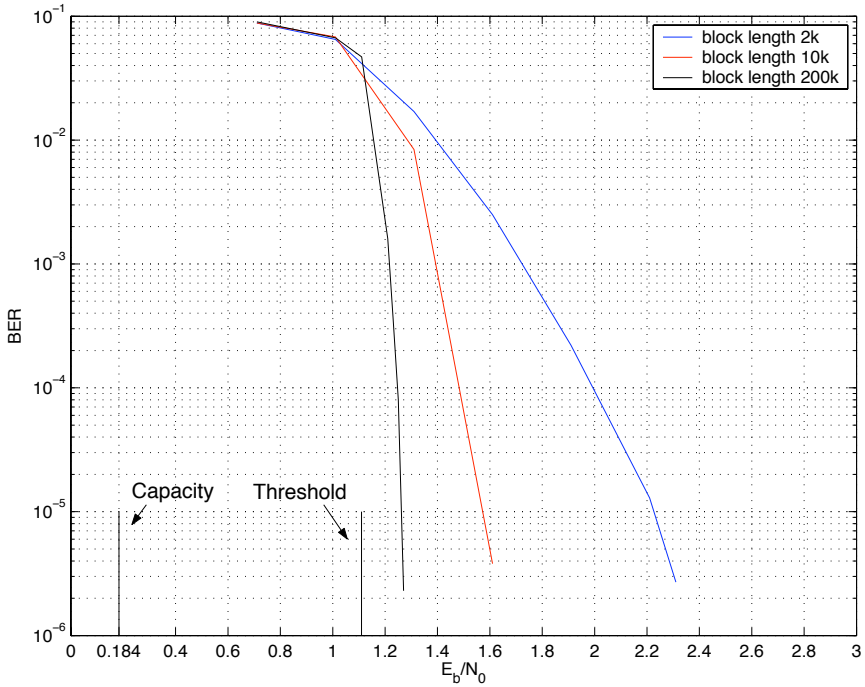
- Assumptions:
  - Certain symmetry conditions are met that allow us to assume the all $+1$ codeword is transmitted.
  - The graph is cycle-free - i.e., it corresponds to an asymptotically long code with large (infinite) girth.
    - So we can assume each message that arrives at a node is a random variable that is independent of every other message arriving at that node.
    - We can use Fourier and Laplace transform methods to compute a closed form expression for the PDF's of the messages passed during the $(\ell+1)^{\text{st}}$ iteration based on the PDF's of the messages passed during the $\ell^{\text{th}}$ iteration.

# Fundamental Limits to Message Passing, continued

- Variations on the density evolution theme:
  - "Gaussian Approximation" – assumes that the messages are Gaussian random variables and therefore can be "tracked" via a single parameter.
    - Symmetry conditions imply the mean and variance of the messages are related by $m = \sigma^2/2$.
  - Extrinsic Information Transfer (EXIT) Charts
    - Uses mutual information between codeword bits and the associated messages to quantify how information is improved at every iteration
    - Net result: An information theoretic characterization of each constituent decoder.
  - Histogram-based techniques - estimates the PDF's of the messages through time when a closed form isn't available (e.g., coded modulation).

# Fundamental Limits to Message Passing, continued

**But the Question Remains:** What can we do to move the thresholds associated with LDPC codes closer to channel capacity?




| $(d_c, d_r)$ | Threshold | Capacity |
|---|---|---|
| (3, 4) | 1.00 dB | - 0.79 dB |
| (3, 5) | 0.97 dB | -0.23 dB |
| (3, 6) | 1.11 dB | 0.184 dB |

**Answer:** Consider irregular LDPC codes.

# Irregular LDPC Codes and Approaching Capacity

**Some Intuition:**

- The *degree* of a node is the number of edges incident on that node.
- Bit nodes with large degrees collect more information from their adjacent check nodes than bit nodes with small degrees - i.e., they're better protected.
  - For regular LDPC codes, if you fix $w_r$ (= the check node degree) and increase $w_c$ (= the bit node degree) you decrease the rate $R \approx 1-(w_c/w_r)$ and create a more powerful code.
- By optimizing the *degree profile* of the code - i.e., by varying bit and check node degrees (= varying the column weights and row weights of $H$) we might be able to improve performance.

# Irregular LDPC Codes and Approaching Capacity, continued

– Throw in a few powerful bit nodes (with large degree) to get decoding started - but not so many that you reduce the rate.

– Then through iterative decoding these powerful bits will spread their reliability throughout the code.

**Definition:** The *degree profile* of a code is specified by two polynomials:

$$\lambda(x) = \sum_i \lambda_i x^{i-1} \quad \text{and} \quad \rho(x) = \sum_j \rho_j x^{j-1},$$

where $\lambda_i$ is the fraction of edges that are incident on degree-$i$ bit nodes and $\rho_j$ is the fraction of edges that are incident on degree-$j$ check nodes.

(Obviously, a $(w_c, w_r)$-regular code has $\lambda(x) = x^{w_c-1}$ and $\rho(x) = x^{w_r-1}$.)

# Irregular LDPC Codes and Approaching Capacity, continued

**Fact:** The rate of an LDPC code with degree profile $[\lambda(x), \rho(x)]$ is bounded by

$$R \geq 1 - \frac{\int_0^1 \rho(x)dx}{\int_0^1 \lambda(x)dx} \left( = 1 - \frac{\frac{1}{w_r}x^{w_r}|_0^1}{\frac{1}{w_c}x^{w_c}|_0^1} = 1 - \frac{w_c}{w_r} \text{ (regular)} \right)$$

with equality if and only if the rows of the parity check matrix are linearly independent.

**Finally:** Given a degree profile $[\lambda(x), \rho(x)]$, we can carry out density evolution in a method similar to that used for regular LDPC codes - and thereby find a threshold associated with $[\lambda(x), \rho(x)]$.

- Now the degree of a node is a random variable with distribution specified by $[\lambda(x), \rho(x)]$ - and that must be taken into account when computing the PDF's of the messages from one iteration to the next.

# Irregular LDPC Codes and Approaching Capacity, continued

**Observation:** Using optimization techniques (e.g., linear programming) we can optimize the degree profile of an LDPC code to produce the best threshold subject to some constraint(s) - constraints such as a rate constraint or a constraint on the maximum degree of $\lambda(x)$ and/or $\rho(x)$.

**Significance:**

- This approach can be used to find good (indeed, optimal) degree profiles - good in the sense that they have low thresholds.
- When codes with these thresholds are (randomly) constructed, the resulting performance tends to have a much better "waterfall region" but may suffer from an "error floor" (which regular LDPC codes *do not have*) due to the presence of degree-2 bit nodes.

# Examples of LDPC Codes

**Source:** "On the Design of Low-Density Parity Check Codes within 0.0045 dB of the Shannon Limit," by Chung, Forney, Richardson, and Urbanke, *IEEE Communications Letters,* February 2001.

**Overview:**

- Uses "discretized density evolution" to find a good degree profile for a rate-$1/2$ code for the BPSK-modulated AWGN channel.
- Optimizes degree profiles subject to these constraints:
  - the "left degree" profile $\lambda(x)$ satisfies $\deg[\lambda(x)] < d_\ell$;
  - the "right degree" profile $\rho(x)$ is "concentrated" on two adjacent values - i.e., $\rho(x) = (1-\rho)x^{j-1} + \rho x^j$ for some integer $j \geq 2$.

# Examples of LDPC Codes, continued

- Results:
  - The threshold for $d_\ell = 8000$ is 0.0045 dB from Shannon limit.
  - The required SNR for a blocklength-$10^7$ code constructed using the $d_\ell = 200$ profile is 0.04 dB from the Shannon limit at a BER of $10^{-6}$.

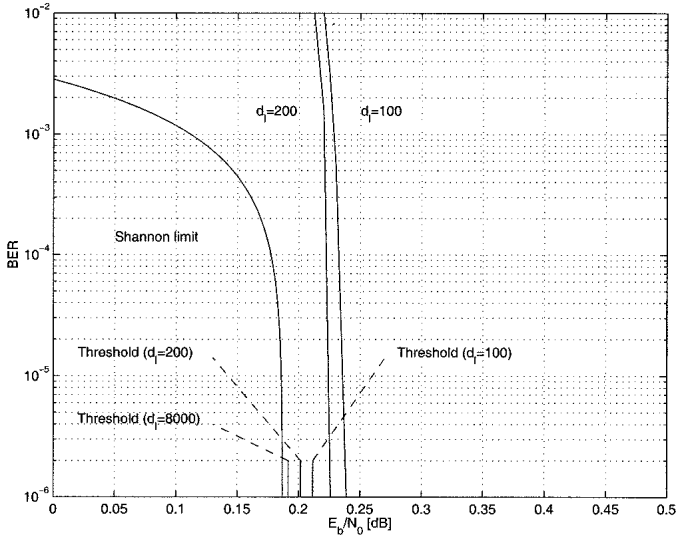



Figure 9: BER performance of optimized rate-1/2 LDPC codes with $d_\ell = 100$ and $d_\ell = 200$ along with associated thresholds and Shannon capacity.

# Examples of LDPC Codes, continued

## LDPC Code for Digital Video Broadcast (DVB-S2)

- "Second generation" DVB for satellite transmission.
- Employs two frame sizes:
  - "Normal" frame of 64,800 bits for delay-insensitive applications.
  - "Short" frame of 16,200 bits for delay-sensitive applications.
- Employs a serial concatenated code structure.
  - Data is first encoded using a BCH code.
    - Capable of up to $t = 12$ error correction.
    - Blocklength varies from $n_{\mathrm{BCH}} = 3,240$ to $n_{\mathrm{BCH}} = 58,320$.
  - Data is then encoded using an LDPC code.
    - Rates vary from $R_{\mathrm{LDPC}} = 1/4$ to $R_{\mathrm{LDPC}} = 9/10$.
    - Blocklength equals the frame length - i.e., either 64,800 or 16,200.

# Examples of LDPC Codes, continued

- Introduces algebraic structure into the connections so as to speed up information flow through the decoder. Specifically:
  - Fix $M = 360$. Then for a group of $M$ bit nodes, if the check nodes connected to the *first* bit node of degree $d_v$ are numbered $[c_1, c_2, \ldots, c_{d_v}]$, then the check nodes connected to the $i^{\text{th}}$ bit node of degree $d_v$ (where $i \leq M$) are numbered

$$[(c_1 + (i-1)q) \bmod R, \quad \ldots, \quad (c_{d_v} + (i-1)q) \bmod R]$$

    where $R$ is the number of check nodes and $q = R/M$.
  - So specifying the connections of just one bit node specifies the connections of $M$ bits nodes - and it facilitates parallel data flow.

# Examples of LDPC Codes, continued

**Table 5a: coding parameters (for normal FECFRAME $n_{ldpc}$ = 64 800)**

| LDPC code | BCH Uncoded Block $K_{bch}$ | BCH coded block $N_{bch}$ LDPC Uncoded Block $k_{ldpc}$ | BCH t-error correction | LDPC Coded Block $n_{ldpc}$ |
|---|---|---|---|---|
| 1/4 | 16 008 | 16 200 | 12 | 64 800 |
| 1/3 | 21 408 | 21 600 | 12 | 64 800 |
| 2/5 | 25 728 | 25 920 | 12 | 64 800 |
| 1/2 | 32 208 | 32 400 | 12 | 64 800 |
| 3/5 | 38 688 | 38 880 | 12 | 64 800 |
| 2/3 | 43 040 | 43 200 | 10 | 64 800 |
| 3/4 | 48 408 | 48 600 | 12 | 64 800 |
| 4/5 | 51 648 | 51 840 | 12 | 64 800 |
| 5/6 | 53 840 | 54 000 | 10 | 64 800 |
| 8/9 | 57 472 | 57 600 | 8 | 64 800 |
| 9/10 | 58 192 | 58 320 | 8 | 64 800 |

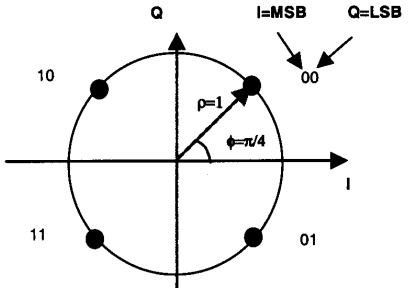
**Table 5b: coding parameters (for short FECFRAME $n_{ldpc}$ = 16 200)**

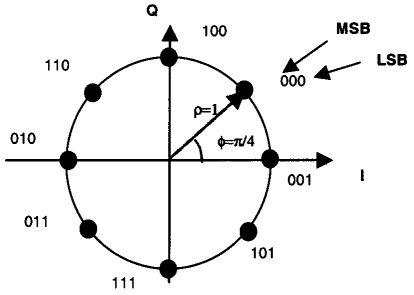
| LDPC Code Identifier | BCH Uncoded Block $K_{bch}$ | BCH coded block $N_{bch}$ LDPC Uncoded Block $k_{ldpc}$ | BCH t-error correction | Effective LDPC Rate $k_{ldpc}$/16 200 | LDPC Coded Block $n_{ldpc}$ |
|---|---|---|---|---|---|
| 1/4 | 3 072 | 3 240 | 12 | 1/5 | 16 200 |
| 1/3 | 5 232 | 5 400 | 12 | 1/3 | 16 200 |
| 2/5 | 6 312 | 6 480 | 12 | 2/5 | 16 200 |
| 1/2 | 7 032 | 7 200 | 12 | 4/9 | 16 200 |
| 3/5 | 9 552 | 9 720 | 12 | 3/5 | 16 200 |
| 2/3 | 10 632 | 10 800 | 12 | 2/3 | 16 200 |
| 3/4 | 11 712 | 11 880 | 12 | 11/15 | 16 200 |
| 4/5 | 12 432 | 12 600 | 12 | 7/9 | 16 200 |
| 5/6 | 13 152 | 13 320 | 12 | 37/45 | 16 200 |
| 8/9 | 14 232 | 14 400 | 12 | 8/9 | 16 200 |
| 9/10 | NA | NA | NA | NA | NA |

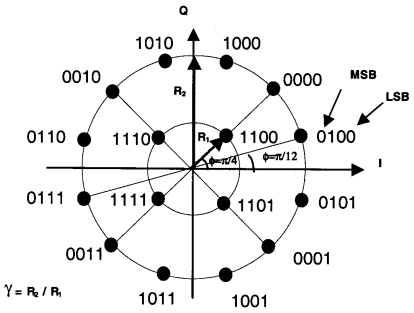
# Examples of LDPC Codes, continued

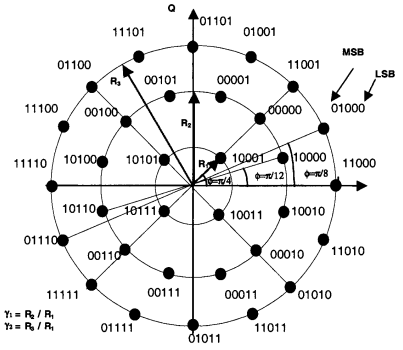
## Modulation:

- Four signal sets - QPSK, 8-PSK, 16-APSK, and 32-APSK.
- Output of LDPC encoder is interleaved and then Gray-mapped onto an appropriate signal constellation.
- Choice of code rate and constellation can adapt the bandwidth efficiency based on application and channel conditions - from 0.5 bps/Hz to 4.5 bps/Hz.



# Examples of LDPC Codes, continued

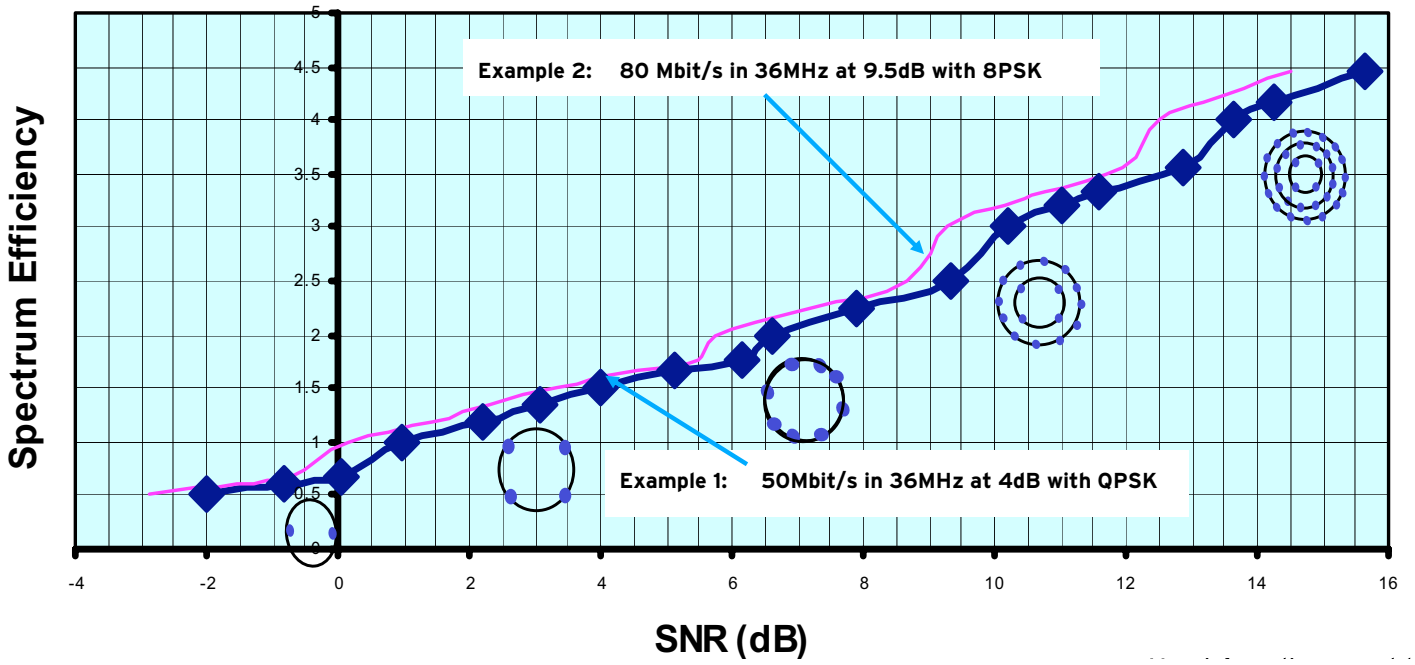



Figure 10: Spectral efficiencies available in DVB-S2.

# Examples of LDPC Codes, continued

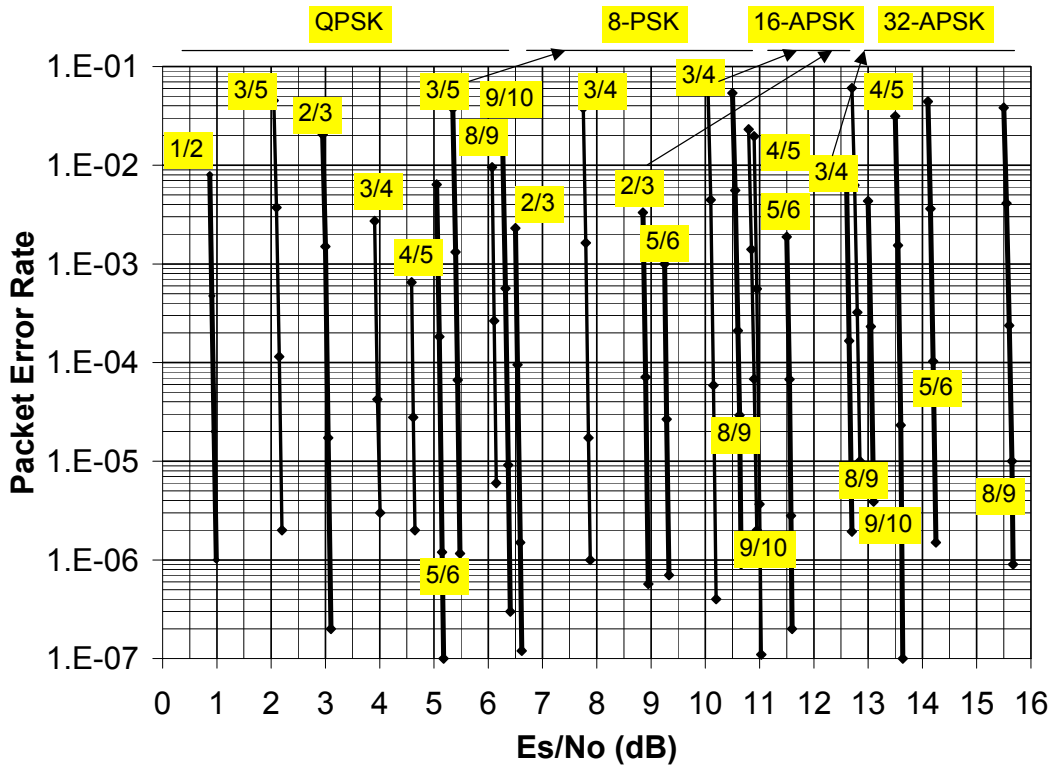



Figure 11: Frame error rate of LDPC codes + BCH codes over an AWGN channel with a frame size of 64,800 bits.

# Examples of LDPC Codes, continued

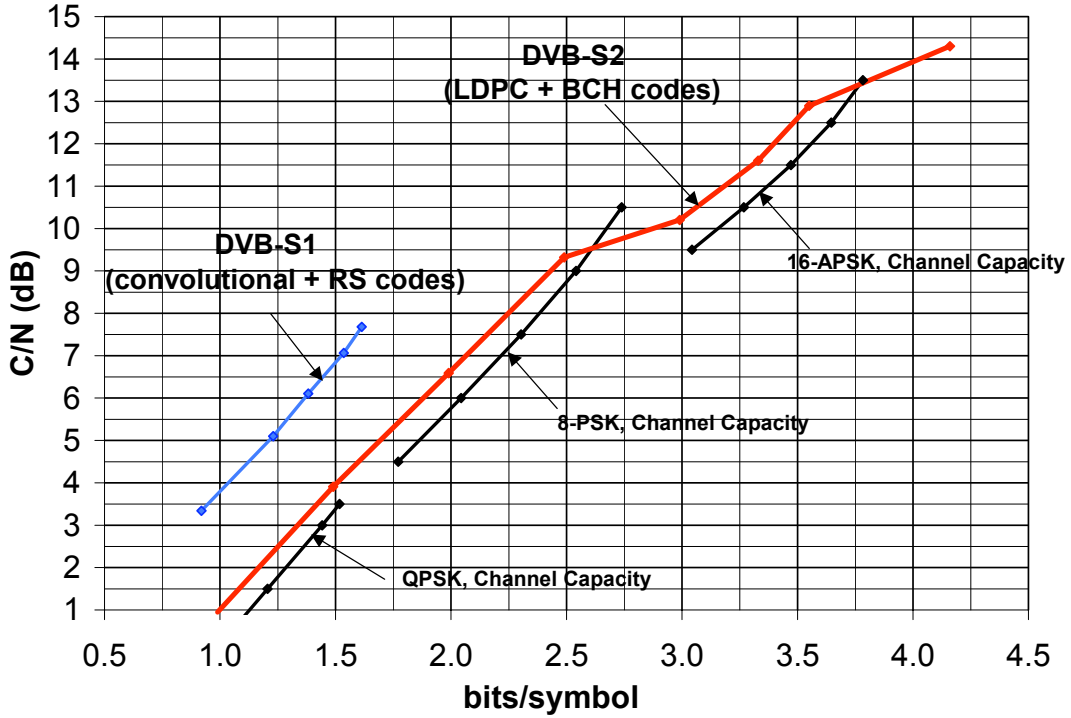



Figure 12: Carrier SNR required to attain an MPEG packet error rate of $10^{-7}$.

# Examples of LDPC Codes, continued

## DVB LDPC Code Characteristics

- Irregular LDPC codes are used, with bit node degrees as indicated in the table below.

| Code Rate | 13 | 12 | 11 | 8 | 4 | 3 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|
| 1/4 | | 5400 | | | | 10800 | 48599 | 1 |
| 1/3 | | 7200 | | | | 14400 | 43199 | 1 |
| 1/2 | | | | 12960 | | 19440 | 32399 | 1 |
| 3/5 | | 12960 | | | | 25920 | 25919 | 1 |
| 2/3 | 4320 | | | | | 38880 | 21599 | 1 |
| 3/4 | | 5400 | | | | 43200 | 16199 | 1 |
| 4/5 | | | 6480 | | | 45360 | 12959 | 1 |
| 5/6 | 5400 | | | | | 48600 | 10799 | 1 |
| 8/9 | | | | | 7200 | 50400 | 7199 | 1 |
| 9/10 | | | | | 6480 | 51840 | 6479 | 1 |

Figure 13: Distribution of bit node degrees in the DVB LDPC code.