

Introduction to LDPC Codes and Factor Graphs

Henry D. Pfister

Introduction to Error Correction Codes
Duke University
Spring 2022

Outline

Low-Density Parity-Check Codes

Factor Graphs

Message Passing

LDPC Ensembles

Outline

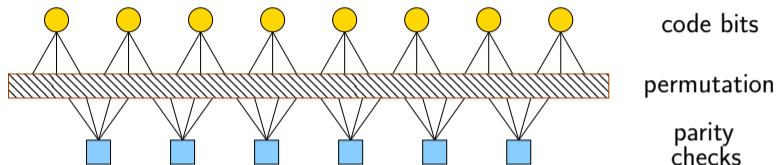
Low-Density Parity-Check Codes

Factor Graphs

Message Passing

LDPC Ensembles

Low-Density Parity-Check (LDPC) Codes



- ▶ Linear codes with a sparse parity-check matrix H
 - ▶ Edge in Tanner graph connects check node i to bit node j if $H_{ij} = 1$
 - ▶ **Circles** represent a code bit value participating in parity checks
 - ▶ **Squares** indicate an even-parity check (i.e., attached bits sum to 0 mod 2)
 - ▶ Naturally leads to **message-passing iterative** (MPI) decoding
 - ▶ Introduced by Gallager in 1960; largely forgotten until 1995

Decoding LDPC Codes

- ▶ Maximum A Posteriori (MAP) Decoder
 - ▶ Optimum decoder that chooses the **most likely codeword**
 - ▶ **Infeasible in practice** due to enormous number of codewords
- ▶ Belief-Propagation (BP) Decoder
 - ▶ Low-complexity message-passing decoder by Gallager
 - ▶ Probability estimates are passed along edges in the Tanner graph
 - ▶ Updates based on assuming **incoming estimates are independent**
- ▶ Density Evolution (DE)
 - ▶ Tracks **distribution of messages** during iterative decoding
 - ▶ BP noise threshold can be **computed via DE**
 - ▶ Long codes decode almost surely if DE predicts success

A Little History

Robert Gallager



introduced LDPC codes in 1962 paper

1962

IRE TRANSACTIONS ON INFORMATION THEORY

21

Low-Density Parity-Check Codes*

R. G. GALLAGER†

Summary—A low-density parity-check code is a code specified by a parity-check matrix with the following properties: each column contains a small fixed number $j \geq 3$ of 1's and each row contains a small fixed number $k > j$ of 1's. The typical minimum distance of these codes increases linearly with block length for a fixed rate and fixed j . When used with maximum likelihood decoding on a sufficiently quiet binary-input symmetric channel, the typical probability of decoding error decreases exponentially with block length for a fixed rate and fixed j .

A simple but nonoptimum decoding scheme operating directly from the channel a posteriori probabilities is described. Both the

equations. We call the set of digits contained in a parity-check equation a parity-check set. For example, the first parity-check set in Fig. 1 is the set of digits (1, 2, 3, 5).

The use of parity-check codes makes coding (as distinguished from decoding) relatively simple to implement. Also, as Elias [3] has shown, if a typical parity-check code of long block length is used on a binary symmetric channel, and if the code rate is between *critical rate* and channel capacity, then the probability of decoding error

Judea Pearl



defined general belief-propagation in 1986 paper

Fusion, Propagation, and Structuring in Belief Networks*

Judea Pearl

Cognitive Systems Laboratory, Computer Science Department,
University of California, Los Angeles, CA 90024, U.S.A.

Recommended by Patrick Hayes

ABSTRACT

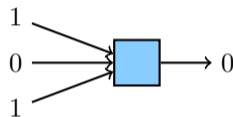
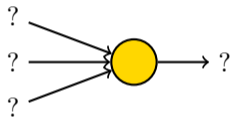
Belief networks are directed acyclic graphs in which the nodes represent propositions (or variables), the arcs signify direct dependencies between the linked propositions, and the strengths of these dependencies are quantified by conditional probabilities. A network of this sort can be used to represent the generic knowledge of a domain expert, and it turns into a computational architecture if the links are used not merely for storing factual knowledge but also for directing and activating the data flow in the computations which manipulate this knowledge.

Message-Passing Decoding

- ▶ Basic Idea
 - ▶ Estimates of bit values are passed along graph edges in alternating phases
 - ▶ bit-to-check phase: bits pass messages to adjacent checks
 - ▶ check-to-bit phase: checks pass messages to adjacent bits
 - ▶ Each **output message depends on other input messages**
- ▶ Simplifications for the binary erasure channel (BEC)
 - ▶ Messages are always **either the correct value or an erasure**
 - ▶ This leads to very simple message combining rules

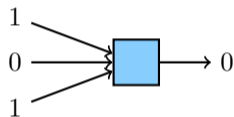
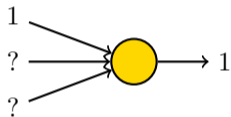
Message-Passing Decoding for the BEC

- ▶ Message passing rules for the BEC
 - ▶ Bits pass an erasure only if all other inputs are erased
 - ▶ Checks pass the correct value if all other inputs are correct and erasure otherwise



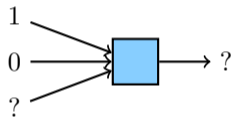
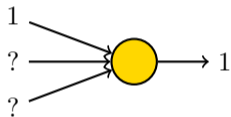
Message-Passing Decoding for the BEC

- ▶ Message passing rules for the BEC
 - ▶ Bits pass an erasure only if all other inputs are erased
 - ▶ Checks pass the correct value if all other inputs are correct and erasure otherwise



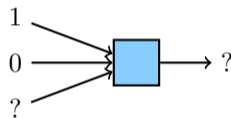
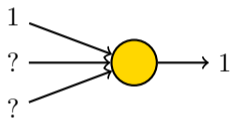
Message-Passing Decoding for the BEC

- ▶ Message passing rules for the BEC
 - ▶ Bits pass an erasure only if all other inputs are erased
 - ▶ Checks pass the correct value if all other inputs are correct and erasure otherwise

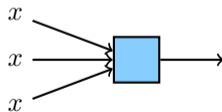
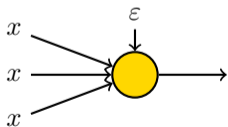


Message-Passing Decoding for the BEC

- ▶ Message passing rules for the BEC
 - ▶ Bits pass an erasure only if all other inputs are erased
 - ▶ Checks pass the correct value if all other inputs are correct and erasure otherwise

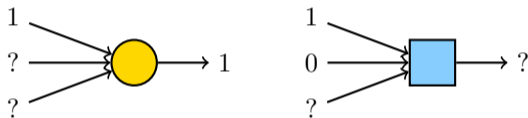


- ▶ If input messages are independently erased with prob. x

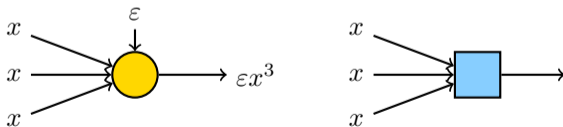


Message-Passing Decoding for the BEC

- ▶ Message passing rules for the BEC
 - ▶ Bits pass an erasure only if all other inputs are erased
 - ▶ Checks pass the correct value if all other inputs are correct and erasure otherwise

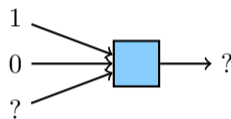
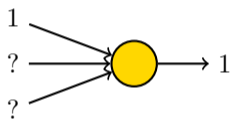


- ▶ If input messages are independently erased with prob. x

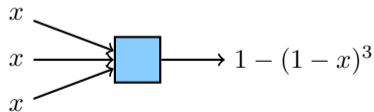
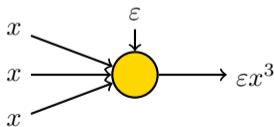


Message-Passing Decoding for the BEC

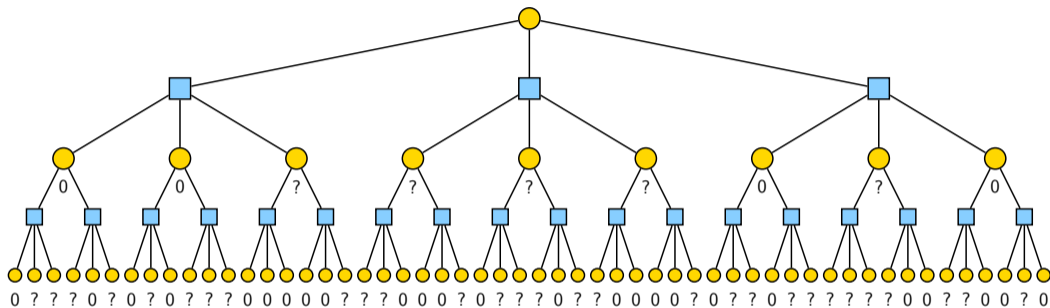
- ▶ Message passing rules for the BEC
 - ▶ Bits pass an erasure only if all other inputs are erased
 - ▶ Checks pass the correct value if all other inputs are correct and erasure otherwise



- ▶ If input messages are independently erased with prob. x

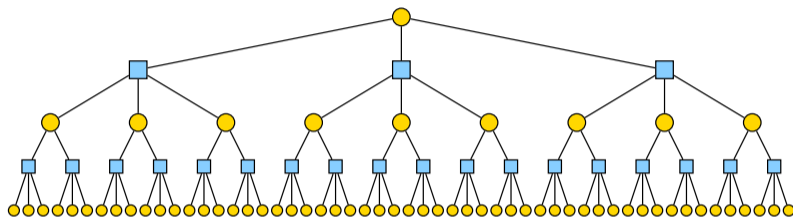


Computation Graph



- ▶ Computation graph for a (3,4)-regular LDPC code
 - ▶ Illustrates decoding from the [perspective of a single bit-node](#)
 - ▶ Show the truncated depth-4 neighborhood of a single bit node
 - ▶ This is sufficient to determine 2 full iterations of message passing
 - ▶ For long random LDPC codes, the graph is typically a tree

Computation Graph for the Message-Passing Decoder



$$\tilde{x}_3 = \varepsilon y_2^3$$

$$y_2 = 1 - (1 - x_2)^3$$

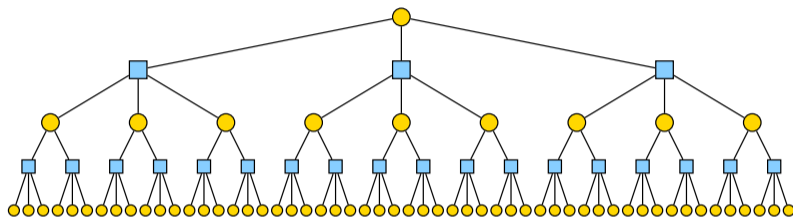
$$x_2 = \varepsilon y_1^2$$

$$y_1 = 1 - (1 - x_1)^3$$

$$x_1 = \varepsilon$$

- ▶ Density evolution evaluates failure probability for a random instance of decoding
 - ▶ Messages are now seen as random variables (either 0 or ?)
 - ▶ Independence at each level follows from tree structure
 - ▶ Allows density evolution to **track message erasure probability**
 - ▶ During i -th iteration, bit-to-check / check-to-bit message erasure prob is x_i / y_i

Computation Graph for the Message-Passing Decoder



$$\tilde{x}_3 = \varepsilon y_2^3$$

$$y_2 = 1 - (1 - x_2)^3$$

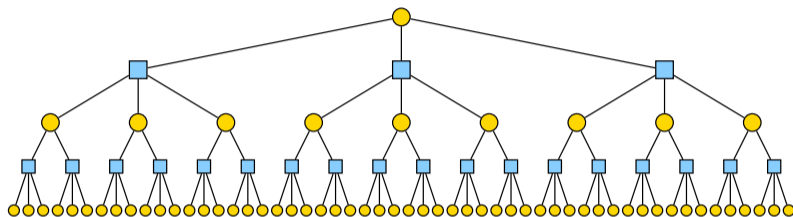
$$x_2 = \varepsilon y_1^2$$

$$y_1 = 1 - (1 - x_1)^3$$

$$x_1 = \varepsilon = 0.600$$

- ▶ Density evolution evaluates failure probability for a random instance of decoding
 - ▶ Messages are now seen as random variables (either 0 or ?)
 - ▶ Independence at each level follows from tree structure
 - ▶ Allows density evolution to **track message erasure probability**
 - ▶ During i -th iteration, bit-to-check / check-to-bit message erasure prob is x_i / y_i

Computation Graph for the Message-Passing Decoder



$$\tilde{x}_3 = \varepsilon y_2^3$$

$$y_2 = 1 - (1 - x_2)^3$$

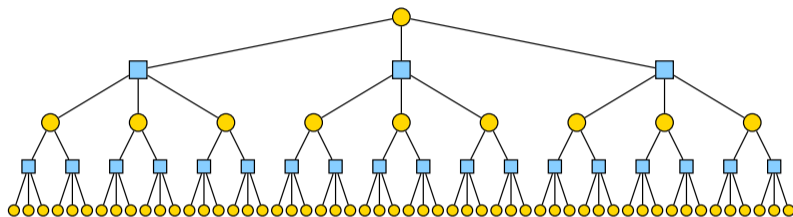
$$x_2 = \varepsilon y_1^2$$

$$y_1 = 1 - (1 - x_1)^3 = 0.936$$

$$x_1 = 0.600$$

- ▶ Density evolution evaluates failure probability for a random instance of decoding
 - ▶ Messages are now seen as random variables (either 0 or ?)
 - ▶ Independence at each level follows from tree structure
 - ▶ Allows density evolution to **track message erasure probability**
 - ▶ During i -th iteration, bit-to-check / check-to-bit message erasure prob is x_i / y_i

Computation Graph for the Message-Passing Decoder



$$\tilde{x}_3 = \varepsilon y_2^3$$

$$y_2 = 1 - (1 - x_2)^3$$

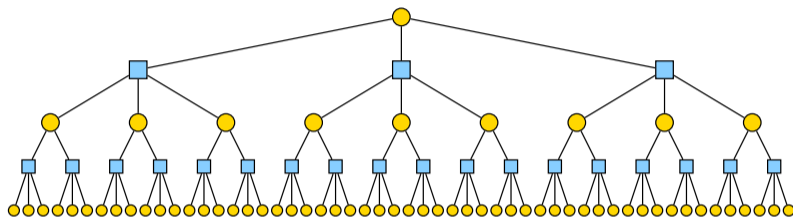
$$x_2 = \varepsilon y_1^2 = 0.526$$

$$y_1 = 0.936$$

$$x_1 = 0.600$$

- ▶ Density evolution evaluates failure probability for a random instance of decoding
 - ▶ Messages are now seen as random variables (either 0 or ?)
 - ▶ Independence at each level follows from tree structure
 - ▶ Allows density evolution to **track message erasure probability**
 - ▶ During i -th iteration, bit-to-check / check-to-bit message erasure prob is x_i / y_i

Computation Graph for the Message-Passing Decoder



$$\tilde{x}_3 = \varepsilon y_2^3$$

$$y_2 = 1 - (1 - x_2)^3 = 0.894$$

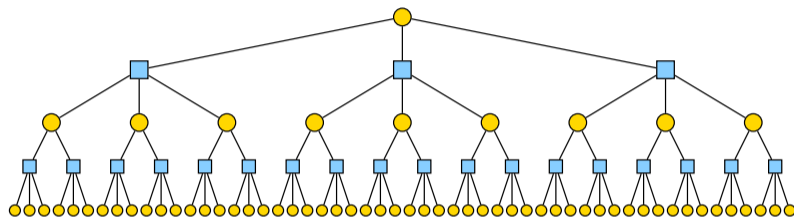
$$x_2 = 0.526$$

$$y_1 = 0.936$$

$$x_1 = 0.600$$

- ▶ Density evolution evaluates failure probability for a random instance of decoding
 - ▶ Messages are now seen as random variables (either 0 or ?)
 - ▶ Independence at each level follows from tree structure
 - ▶ Allows density evolution to **track message erasure probability**
 - ▶ During i -th iteration, bit-to-check / check-to-bit message erasure prob is x_i / y_i

Computation Graph for the Message-Passing Decoder



$$\tilde{x}_3 = \varepsilon y_2^3 = 0.429$$

$$y_2 = 0.894$$

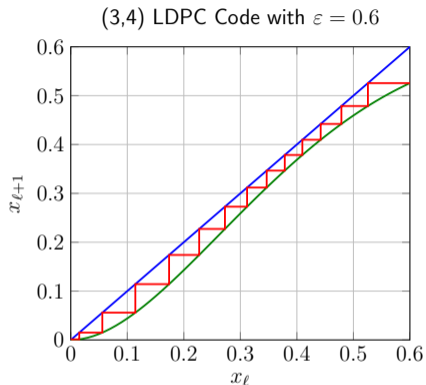
$$x_2 = 0.526$$

$$y_1 = 0.936$$

$$x_1 = 0.600$$

- ▶ Density evolution evaluates failure probability for a random instance of decoding
 - ▶ Messages are now seen as random variables (either 0 or ?)
 - ▶ Independence at each level follows from tree structure
 - ▶ Allows density evolution to **track message erasure probability**
 - ▶ During i -th iteration, bit-to-check / check-to-bit message erasure prob is x_i / y_i

Density Evolution (DE) for LDPC Codes



Density evolution for a (3, 4)-regular LDPC code:

$$x_{\ell+1} = \varepsilon (1 - (1 - x_\ell)^3)^2$$

Decoding Thresholds:

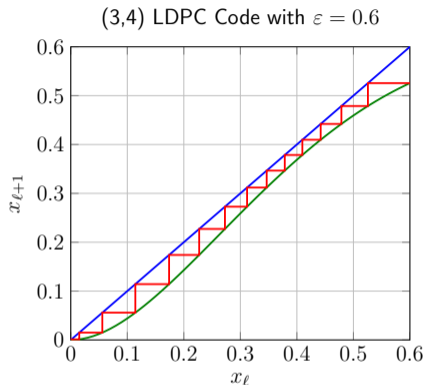
$$\varepsilon^{\text{BP}} \approx 0.647$$

$$\varepsilon^{\text{MAP}} \approx 0.746$$

$$\varepsilon^{\text{Sh}} = 0.750$$

- DE tracks bit-to-check msg erasure rate x_ℓ after ℓ iterations

Density Evolution (DE) for LDPC Codes



Density evolution for a (3, 4)-regular LDPC code:

$$x_{\ell+1} = \varepsilon (1 - (1 - x_\ell)^3)^2$$

Decoding Thresholds:

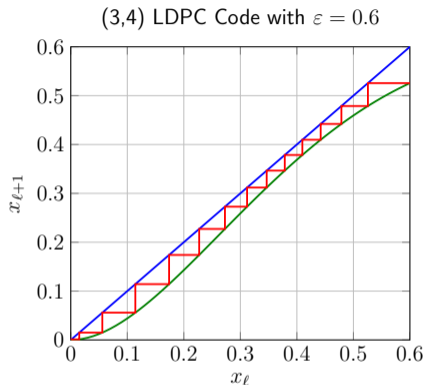
$$\varepsilon^{\text{BP}} \approx 0.647$$

$$\varepsilon^{\text{MAP}} \approx 0.746$$

$$\varepsilon^{\text{Sh}} = 0.750$$

- ▶ DE tracks bit-to-check msg erasure rate x_ℓ after ℓ iterations
- ▶ x_ℓ decreases to a limit $x_\infty(\varepsilon)$ that depends on ε

Density Evolution (DE) for LDPC Codes



Density evolution for a (3, 4)-regular LDPC code:

$$x_{\ell+1} = \varepsilon (1 - (1 - x_\ell)^3)^2$$

Decoding Thresholds:

$$\varepsilon^{\text{BP}} \approx 0.647$$

$$\varepsilon^{\text{MAP}} \approx 0.746$$

$$\varepsilon^{\text{Sh}} = 0.750$$

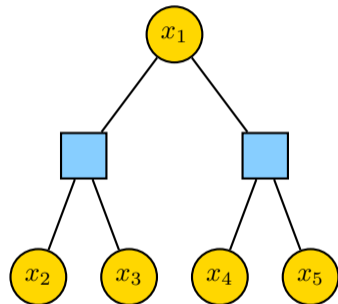
- ▶ DE tracks bit-to-check msg erasure rate x_ℓ after ℓ iterations
- ▶ x_ℓ decreases to a limit $x_\infty(\varepsilon)$ that depends on ε
- ▶ As $n \rightarrow \infty$, decoding succeeds if ε less than the BP noise threshold
 - ▶ $\varepsilon^{\text{BP}} = \sup\{\varepsilon \in [0, 1] \mid x_\infty(\varepsilon) = 0\}$ (easily computed numerically)

LDPC Decoding on General Channels

- ▶ Consider a binary-input DMC defined by $W(y|x)$ with $x \in \{0, 1\}$ and $y \in \mathcal{Y}$
- ▶ Sufficient Statistics
 - ▶ A statistic $T(y)$ is **sufficient** for the input if $Y - T(Y) - X$ forms a Markov chain
 - ▶ The conditional probability (i.e., $T(y) \triangleq \mathbb{P}(X = 1|Y = y)$) is always sufficient
- ▶ Channel Combining Rules
 - ▶ A tree Tanner graph defines a combined channel whose input x is the root bit and whose output y is the combination of all bit-node observations in the tree
 - ▶ The output space becomes quite complicated, so we compress to a sufficient statistic
 - ▶ One can get efficient decoding by propagating sufficient statistics up the tree

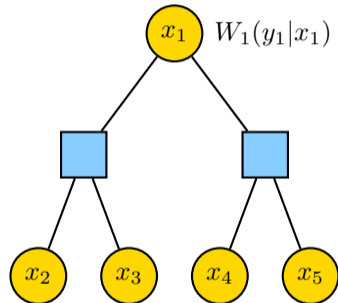
Channel Combining

- ▶ 5 bits satisfying 2 parity checks



Channel Combining

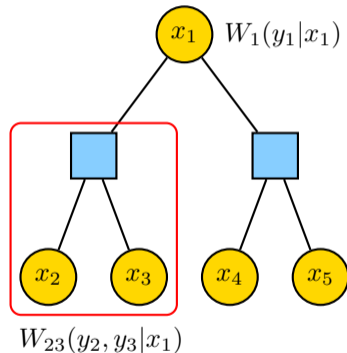
- ▶ 5 bits satisfying 2 parity checks
- ▶ Bit- i observed directly through channel $W_i(y_i|x_i)$



Channel Combining

- ▶ 5 bits satisfying 2 parity checks
- ▶ Bit- i observed directly through channel $W_i(y_i|x_i)$
- ▶ Treat y_2, y_3 as output of combined channel W_{23} :

$$W_{23}(y_2, y_3|x_1) = \frac{1}{2} \sum_{u \in \{0,1\}} W_2(y_2|x_1 \oplus u) W_3(y_3|u)$$

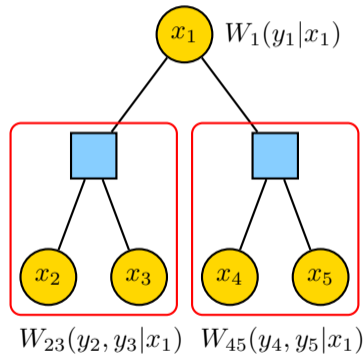


Channel Combining

- ▶ 5 bits satisfying 2 parity checks
- ▶ Bit- i observed directly through channel $W_i(y_i|x_i)$
- ▶ Treat y_2, y_3 as output of combined channel W_{23} :

$$W_{23}(y_2, y_3|x_1) = \frac{1}{2} \sum_{u \in \{0,1\}} W_2(y_2|x_1 \oplus u) W_3(y_3|u)$$

- ▶ This is called **check combining** and may also be applied to treat y_4, y_5 as the output of W_{45}



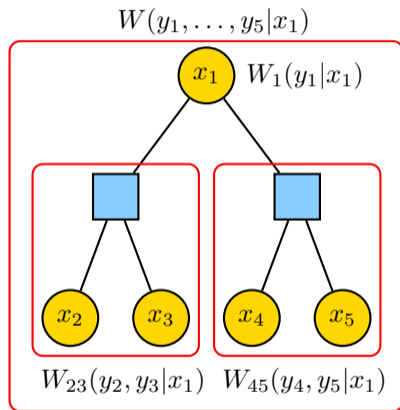
Channel Combining

- ▶ 5 bits satisfying 2 parity checks
- ▶ Bit- i observed directly through channel $W_i(y_i|x_i)$
- ▶ Treat y_2, y_3 as output of combined channel W_{23} :

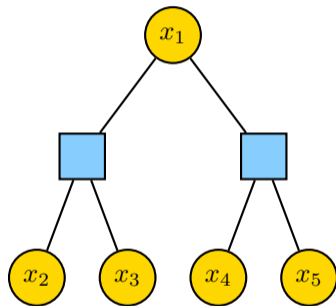
$$W_{23}(y_2, y_3|x_1) = \frac{1}{2} \sum_{u \in \{0,1\}} W_2(y_2|x_1 \oplus u) W_3(y_3|u)$$

- ▶ This is called **check combining** and may also be applied to treat y_4, y_5 as the output of W_{45}
- ▶ The overall channel for x_1 is defined by grouping independent observations (called **bit combining**):

$$W(y_1, \dots, y_5|x_1) = W_1(y_1|x_1)W_{23}(y_2, y_3|x_1)W_{45}(y_4, y_5|x_1)$$



Channel Combining Example



Outline

Low-Density Parity-Check Codes

Factor Graphs

Message Passing

LDPC Ensembles

Factor Graphs

- ▶ A factor graph provides a **graphical representation** of the **local dependence structure** for a set of random variables
 - ▶ Bipartite graph with variables x_1, \dots, x_n and factors f_1, \dots, f_m

Factor Graphs

- ▶ A factor graph provides a **graphical representation** of the **local dependence structure** for a set of random variables
 - ▶ Bipartite graph with variables x_1, \dots, x_n and factors f_1, \dots, f_m
- ▶ Consider random variables $(X_1, X_2, \dots, X_4) \in \mathcal{X}^4$ and Y where:

$$\begin{aligned} P(x_1, x_2, x_3, x_4) &\triangleq \mathbb{P}(X_1 = x_1, X_2 = x_2, \dots, X_4 = x_4 | Y = y) \\ &\propto f(x_1, x_2, x_3, x_4) \\ &\triangleq f_1(x_1, x_2) f_2(x_2, x_3) f_3(x_3, x_4) \end{aligned}$$

Factor Graphs

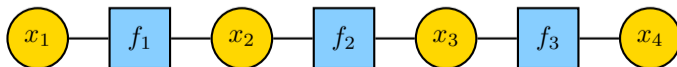
- ▶ A factor graph provides a **graphical representation** of the **local dependence structure** for a set of random variables

- ▶ Bipartite graph with variables x_1, \dots, x_n and factors f_1, \dots, f_m

- ▶ Consider random variables $(X_1, X_2, \dots, X_4) \in \mathcal{X}^4$ and Y where:

$$\begin{aligned} P(x_1, x_2, x_3, x_4) &\triangleq \mathbb{P}(X_1 = x_1, X_2 = x_2, \dots, X_4 = x_4 | Y = y) \\ &\propto f(x_1, x_2, x_3, x_4) \\ &\triangleq f_1(x_1, x_2) f_2(x_2, x_3) f_3(x_3, x_4) \end{aligned}$$

- ▶ Given $Y = y$, this describes a **Markov chain** whose **factor graph** is



Conditional Independence for Factor Graphs

- ▶ Let $A, B, S \subset [n]$ be disjoint subsets of VNs in factor graph G
 - ▶ If S separates A from B (i.e., there is no path in G from A to B that avoids S), then we have $X_A \perp\!\!\!\perp X_B \mid X_S$

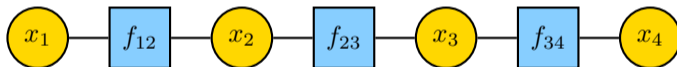
$$P(x_A, x_B | x_S) = P(x_A | x_S)P(x_B | x_S)$$

Conditional Independence for Factor Graphs

- ▶ Let $A, B, S \subset [n]$ be disjoint subsets of VNs in factor graph G
 - ▶ If S separates A from B (i.e., there is no path in G from A to B that avoids S), then we have $X_A \perp\!\!\!\perp X_B \mid X_S$

$$P(x_A, x_B | x_S) = P(x_A | x_S) P(x_B | x_S)$$

- ▶ Markov chain example: $A = \{x_1, x_2\}$, $B = \{x_4\}$, $S = \{x_3\}$

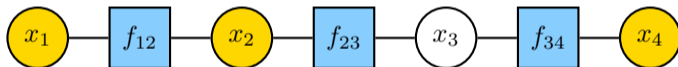


Conditional Independence for Factor Graphs

- ▶ Let $A, B, S \subset [n]$ be disjoint subsets of VNs in factor graph G
 - ▶ If S separates A from B (i.e., there is no path in G from A to B that avoids S), then we have $X_A \perp\!\!\!\perp X_B \mid X_S$

$$P(x_A, x_B | x_S) = P(x_A | x_S) P(x_B | x_S)$$

- ▶ Markov chain example: $A = \{x_1, x_2\}$, $B = \{x_4\}$, $S = \{x_3\}$



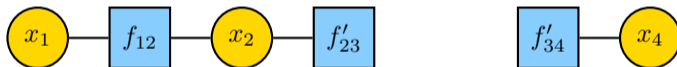
- ▶ Sketch of Proof:
 - ▶ Fixing $X_S = x_S$ separates the FG into disjoint components

Conditional Independence for Factor Graphs

- ▶ Let $A, B, S \subset [n]$ be disjoint subsets of VNs in factor graph G
 - ▶ If S separates A from B (i.e., there is no path in G from A to B that avoids S), then we have $X_A \perp\!\!\!\perp X_B \mid X_S$

$$P(x_A, x_B \mid x_S) = P(x_A \mid x_S)P(x_B \mid x_S)$$

- ▶ Markov chain example: $A = \{x_1, x_2\}$, $B = \{x_4\}$, $S = \{x_3\}$



- ▶ Sketch of Proof:
 - ▶ Fixing $X_S = x_S$ separates the FG into disjoint components
 - ▶ Groups of VNs in different components are independent
 - ▶ $X_A \perp\!\!\!\perp X_B$ because A and B are in different components

Inference via Marginalization

- ▶ Marginalizing out all variables except X_1 gives

$$\mathbb{P}(X_1 = x_1 | Y = y) \propto g_1(x_1) \triangleq \sum_{(x_2, \dots, x_4) \in \mathcal{X}^3} f(x_1, x_2, x_3, x_4)$$

- ▶ Thus, the maximum a posteriori decision for X_1 given $Y = y$ is

$$\hat{x}_1 = \arg \max_{x_1 \in \mathcal{X}} \sum_{(x_2, \dots, x_4) \in \mathcal{X}^3} f(x_1, x_2, x_3, x_4)$$

- ▶ For a general function, this requires roughly $|\mathcal{X}|^4$ operations

Inference via Marginalization

- ▶ Marginalizing out all variables except X_1 gives

$$\mathbb{P}(X_1 = x_1 | Y = y) \propto g_1(x_1) \triangleq \sum_{(x_2, \dots, x_4) \in \mathcal{X}^3} f(x_1, x_2, x_3, x_4)$$

- ▶ Thus, the maximum a posteriori decision for X_1 given $Y = y$ is

$$\hat{x}_1 = \arg \max_{x_1 \in \mathcal{X}} \sum_{(x_2, \dots, x_4) \in \mathcal{X}^3} f(x_1, x_2, x_3, x_4)$$

- ▶ For a general function, this requires roughly $|\mathcal{X}|^4$ operations
- ▶ Marginalization is efficient for tree-structured factor graphs
 - ▶ For the Markov chain, roughly $5|\mathcal{X}|^2$ operations required

$$g_1(x_1) = \sum_{x_2 \in \mathcal{X}} f_1(x_1, x_2) \sum_{x_3 \in \mathcal{X}} f_2(x_2, x_3) \sum_{x_4 \in \mathcal{X}} f_3(x_3, x_4)$$

The Importance of Factorization (1)

- ▶ Consider a random vector $(X_1, X_2, \dots, X_6) \in \mathcal{X}^6$ where

$$\mathbb{P}(X_1 = x_1, \dots, X_6 = x_6 | Y = y) \propto f(x_1, x_2, x_3, x_4, x_5, x_6)$$

The Importance of Factorization (1)

- ▶ Consider a random vector $(X_1, X_2, \dots, X_6) \in \mathcal{X}^6$ where

$$\mathbb{P}(X_1 = x_1, \dots, X_6 = x_6 | Y = y) \propto f(x_1, x_2, x_3, x_4, x_5, x_6)$$

- ▶ Brute force marginal requires $|\mathcal{X}|^5$ operations for each $x_1 \in \mathcal{X}$:

$$g_1(x_1) \triangleq \sum_{x_2^6 \in \mathcal{X}^5} f(x_1, x_2, x_3, x_4, x_5, x_6)$$

- ▶ Thus, we need $|\mathcal{X}|^6$ operations

The Importance of Factorization (1)

- ▶ Consider a random vector $(X_1, X_2, \dots, X_6) \in \mathcal{X}^6$ where

$$\mathbb{P}(X_1 = x_1, \dots, X_6 = x_6 | Y = y) \propto f(x_1, x_2, x_3, x_4, x_5, x_6)$$

- ▶ Brute force marginal requires $|\mathcal{X}|^5$ operations for each $x_1 \in \mathcal{X}$:

$$g_1(x_1) \triangleq \sum_{x_2 \in \mathcal{X}^5} f(x_1, x_2, x_3, x_4, x_5, x_6)$$

- ▶ Thus, we need $|\mathcal{X}|^6$ operations
- ▶ If f **factor**s as follows, then the marginalization can be simplified:

$$f(x_1, x_2, x_3, x_4, x_5, x_6) = f_1(x_1, x_2, x_3) f_2(x_1, x_4, x_6) f_3(x_4) f_4(x_4, x_5)$$

The Importance of Factorization (2)

For example, we can write $g_1(x_1)$ as:

$$= \sum_{x_2} f_1(x_1, x_2, x_3) f_2(x_1, x_4, x_6) f_3(x_4) f_4(x_4, x_5)$$

The Importance of Factorization (2)

For example, we can write $g_1(x_1)$ as:

$$\begin{aligned} &= \sum_{x_2}^6 f_1(x_1, x_2, x_3) f_2(x_1, x_4, x_6) f_3(x_4) f_4(x_4, x_5) \\ &= \sum_{x_2}^5 f_1(x_1, x_2, x_3) f_3(x_4) f_4(x_4, x_5) \left[\sum_{x_6} f_2(x_1, x_4, x_6) \right] \end{aligned}$$

The Importance of Factorization (2)

For example, we can write $g_1(x_1)$ as:

$$\begin{aligned} &= \sum_{x_2}^6 f_1(x_1, x_2, x_3) f_2(x_1, x_4, x_6) f_3(x_4) f_4(x_4, x_5) \\ &= \sum_{x_2}^5 f_1(x_1, x_2, x_3) f_3(x_4) f_4(x_4, x_5) \left[\sum_{x_6} f_2(x_1, x_4, x_6) \right] \\ &= \sum_{x_2}^4 f_1(x_1, x_2, x_3) f_3(x_4) \left[\sum_{x_5} f_4(x_4, x_5) \right] \left[\sum_{x_6} f_2(x_1, x_4, x_6) \right] \end{aligned}$$

The Importance of Factorization (2)

For example, we can write $g_1(x_1)$ as:

$$\begin{aligned} &= \sum_{x_2}^6 f_1(x_1, x_2, x_3) f_2(x_1, x_4, x_6) f_3(x_4) f_4(x_4, x_5) \\ &= \sum_{x_2}^5 f_1(x_1, x_2, x_3) f_3(x_4) f_4(x_4, x_5) \left[\sum_{x_6} f_2(x_1, x_4, x_6) \right] \\ &= \sum_{x_2}^4 f_1(x_1, x_2, x_3) f_3(x_4) \left[\sum_{x_5} f_4(x_4, x_5) \right] \left[\sum_{x_6} f_2(x_1, x_4, x_6) \right] \\ &= \sum_{x_2}^3 f_1(x_1, x_2, x_3) \left[\sum_{x_4} f_3(x_4) \left[\sum_{x_5} f_4(x_4, x_5) \right] \left[\sum_{x_6} f_2(x_1, x_4, x_6) \right] \right] \end{aligned}$$

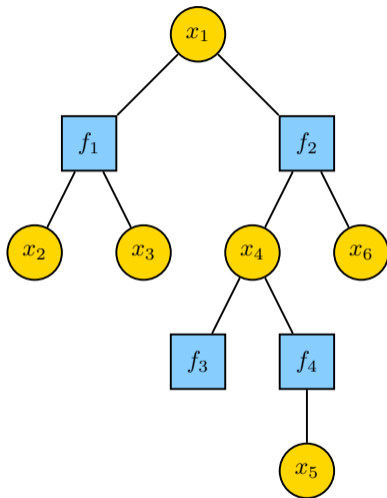
The Importance of Factorization (2)

For example, we can write $g_1(x_1)$ as:

$$\begin{aligned} &= \sum_{x_2}^6 f_1(x_1, x_2, x_3) f_2(x_1, x_4, x_6) f_3(x_4) f_4(x_4, x_5) \\ &= \sum_{x_2}^5 f_1(x_1, x_2, x_3) f_3(x_4) f_4(x_4, x_5) \left[\sum_{x_6} f_2(x_1, x_4, x_6) \right] \\ &= \sum_{x_2}^4 f_1(x_1, x_2, x_3) f_3(x_4) \left[\sum_{x_5} f_4(x_4, x_5) \right] \left[\sum_{x_6} f_2(x_1, x_4, x_6) \right] \\ &= \sum_{x_2}^3 f_1(x_1, x_2, x_3) \left[\sum_{x_4} f_3(x_4) \left[\sum_{x_5} f_4(x_4, x_5) \right] \left[\sum_{x_6} f_2(x_1, x_4, x_6) \right] \right] \\ &= \sum_{x_2} \left[\sum_{x_3} f_1(x_1, x_2, x_3) \right] \left[\sum_{x_4} f_3(x_4) \left[\sum_{x_5} f_4(x_4, x_5) \right] \left[\sum_{x_6} f_2(x_1, x_4, x_6) \right] \right] \end{aligned}$$

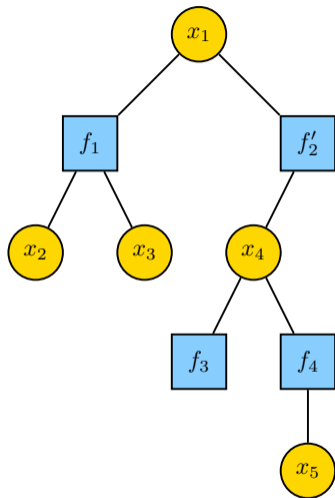
This implementation requires roughly $2|\mathcal{X}|^3 + O(|\mathcal{X}|^2)$ operations

The Factor Graph and Leaf Removal



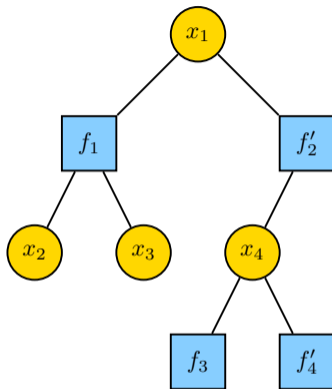
$$g_1(x_1) = \sum_{x_2^5} f_1(x_1, x_2, x_3) f_3(x_4) f_4(x_4, x_5) \sum_{x_6} f_2(x_1, x_4, x_6)$$

The Factor Graph and Leaf Removal



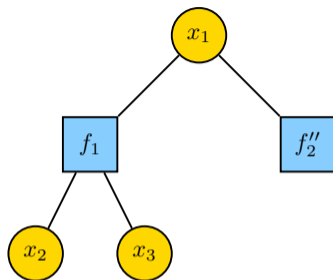
$$g_1(x_1) = \sum_{x_2} f_1(x_1, x_2, x_3) f_3(x_4) \left[\sum_{x_5} f_4(x_4, x_5) \right] f_2'(x_1, x_4)$$

The Factor Graph and Leaf Removal



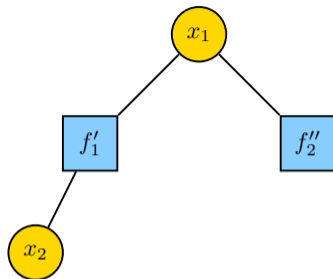
$$g_1(x_1) = \sum_{x_2} f_1(x_1, x_2, x_3) \left[\sum_{x_4} f_3(x_4) f_4'(x_4) f_2'(x_1, x_4) \right]$$

The Factor Graph and Leaf Removal



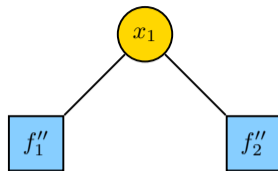
$$g_1(x_1) = \sum_{x_2} \left[\sum_{x_3} f_1(x_1, x_2, x_3) \right] f_2''(x_1)$$

The Factor Graph and Leaf Removal



$$g_1(x_1) = \left[\sum_{x_2} f'_1(x_1, x_2) \right] f'_2(x_1)$$

The Factor Graph and Leaf Removal



$$g_1(x_1) = f_1''(x_1)f_2''(x_1)$$

Factor Graphs and Distributions

- ▶ A non-negative function $f: \mathcal{X}^n \rightarrow \mathbb{R}$ defines a distribution on \mathcal{X}^n :

$$\begin{aligned} P(\underline{x}) &\triangleq \mathbb{P}(X_1 = x_1, \dots, X_n = x_n) \\ &= \frac{1}{Z} f(\underline{x}) \triangleq \frac{1}{Z} \prod_{a=1}^m f_a(\underline{x}_{\partial a}), \end{aligned}$$

- ▶ where $\underline{x}_{\partial a}$ is the subvector of variables involved in factor a
- ▶ and $Z \triangleq \sum_{\underline{x}} f(\underline{x})$ is called the partition function

Factor Graphs and Distributions

- ▶ A non-negative function $f: \mathcal{X}^n \rightarrow \mathbb{R}$ defines a distribution on \mathcal{X}^n :

$$\begin{aligned} P(\underline{x}) &\triangleq \mathbb{P}(X_1 = x_1, \dots, X_n = x_n) \\ &= \frac{1}{Z} f(\underline{x}) \triangleq \frac{1}{Z} \prod_{a=1}^m f_a(\underline{x}_{\partial a}), \end{aligned}$$

- ▶ where $\underline{x}_{\partial a}$ is the subvector of variables involved in factor a
- ▶ and $Z \triangleq \sum_{\underline{x}} f(\underline{x})$ is called the partition function
- ▶ Zero-one factors define constraint satisfaction problems (CSPs)
 - ▶ All factors satisfy $f_a(\underline{x}_{\partial a}) \in \{0, 1\}$ for all $\underline{x}_{\partial a} \in \mathcal{X}^{|\partial a|}$
 - ▶ The set of valid configurations is $\{\underline{x} \in \mathcal{X}^n \mid f(\underline{x}) = 1\}$
 - ▶ Thus, Z equals the number of valid configurations
 - ▶ $P(\underline{x})$ is uniform over the set of valid configurations

Sudoku: A Factor Graph for the Masses

	2		5		1		9	
8			2		3			6
	3			6			7	
		1				6		
5	4						1	9
		2				7		
	9			3			8	
2			8		4			7
	1		9		7		6	

rows are permutations of $\{1, 2, \dots, 9\}$

Sudoku: A Factor Graph for the Masses

	2		5		1		9	
8			2		3			6
	3			6			7	
		1				6		
5	4						1	9
		2				7		
	9			3			8	
2			8		4			7
	1		9		7		6	

rows are permutations of $\{1, 2, \dots, 9\}$
columns are permutations of $\{1, 2, \dots, 9\}$

Sudoku: A Factor Graph for the Masses

	2		5		1		9	
8			2		3			6
	3			6			7	
		1				6		
5	4						1	9
		2				7		
	9			3			8	
2			8		4			7
	1		9		7		6	

rows are permutations of $\{1, 2, \dots, 9\}$

columns are permutations of $\{1, 2, \dots, 9\}$

subblocks are permutations of $\{1, 2, \dots, 9\}$

Sudoku: A Factor Graph for the Masses

	2		5		1		9	
8			2		3			6
3			6				7	
		1				6		
5	4						1	9
		2				7		
	9			3			8	
2			8		4			7
	1		9		7		6	

x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	x_{16}	x_{17}	x_{18}	x_{19}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}	x_{26}	x_{27}	x_{28}	x_{29}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}	x_{36}	x_{37}	x_{38}	x_{39}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}	x_{46}	x_{47}	x_{48}	x_{49}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}	x_{56}	x_{57}	x_{58}	x_{59}
x_{61}	x_{62}	x_{63}	x_{64}	x_{65}	x_{66}	x_{67}	x_{68}	x_{69}
x_{71}	x_{72}	x_{73}	x_{74}	x_{75}	x_{76}	x_{77}	x_{78}	x_{79}
x_{81}	x_{82}	x_{83}	x_{84}	x_{85}	x_{86}	x_{87}	x_{88}	x_{89}
x_{91}	x_{92}	x_{93}	x_{94}	x_{95}	x_{96}	x_{97}	x_{98}	x_{99}

rows are permutations of $\{1, 2, \dots, 9\}$
columns are permutations of $\{1, 2, \dots, 9\}$
subblocks are permutations of $\{1, 2, \dots, 9\}$

implied factor graph has
81 variable and 27 factor nodes

$$f(\underline{x}) = \left(\prod_{i=1}^9 f_{\sigma}(x_{i*}) \right) \left(\prod_{j=1}^9 f_{\sigma}(x_{*j}) \right) \left(\prod_{k=1}^9 f_{\sigma}(x_{B(k)}) \right) \prod_{(i,j) \in O} \mathbb{I}(x_{ij} = y_{ij})$$

Outline

Low-Density Parity-Check Codes

Factor Graphs

Message Passing

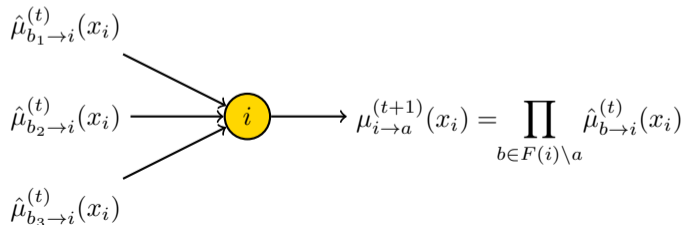
LDPC Ensembles

Marginalization via Belief Propagation

- ▶ Factor Graph $G = (V \cup F, E)$
 - ▶ Variable nodes V , Factor nodes F
 - ▶ Edges: $(i, a) \in E \subseteq V \times F$
 - ▶ $F(i)/V(a) =$ set of neighbors for node- i/a
 - ▶ Messages: $\mu_{i \rightarrow a}^{(t)}(x_i)$ and $\hat{\mu}_{a \rightarrow i}^{(t)}(x_i)$

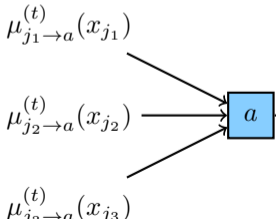
Marginalization via Belief Propagation

- ▶ Factor Graph $G = (V \cup F, E)$
 - ▶ Variable nodes V , Factor nodes F
 - ▶ Edges: $(i, a) \in E \subseteq V \times F$
 - ▶ $F(i)/V(a) =$ set of neighbors for node- i/a
 - ▶ Messages: $\mu_{i \rightarrow a}^{(t)}(x_i)$ and $\hat{\mu}_{a \rightarrow i}^{(t)}(x_i)$
- ▶ variable- i to factor- a message



Marginalization via Belief Propagation

- ▶ Factor Graph $G = (V \cup F, E)$
 - ▶ Variable nodes V , Factor nodes F
 - ▶ Edges: $(i, a) \in E \subseteq V \times F$
 - ▶ $F(i)/V(a) =$ set of neighbors for node- i/a
 - ▶ Messages: $\mu_{i \rightarrow a}^{(t)}(x_i)$ and $\hat{\mu}_{a \rightarrow i}^{(t)}(x_i)$
- ▶ factor- a to variable- i message

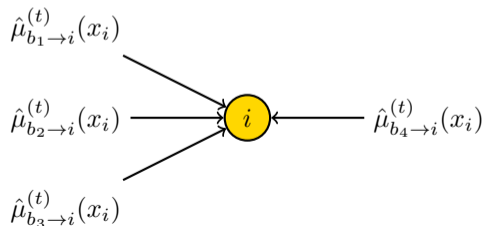


The diagram shows a central blue square node labeled 'a'. Three arrows point towards it from the left, labeled $\mu_{j_1 \rightarrow a}^{(t)}(x_{j_1})$, $\mu_{j_2 \rightarrow a}^{(t)}(x_{j_2})$, and $\mu_{j_3 \rightarrow a}^{(t)}(x_{j_3})$. An arrow points from node 'a' to the right, towards the equation for $\hat{\mu}_{a \rightarrow i}^{(t)}(x_i)$.

$$\hat{\mu}_{a \rightarrow i}^{(t)}(x_i) = \sum_{\underline{x}_{V(a) \setminus i}} f_a(\underline{x}_{V(a)}) \prod_{j \in V(a) \setminus i} \mu_{j \rightarrow a}^{(t)}(x_j)$$

Marginalization via Belief Propagation

- ▶ Factor Graph $G = (V \cup F, E)$
 - ▶ Variable nodes V , Factor nodes F
 - ▶ Edges: $(i, a) \in E \subseteq V \times F$
 - ▶ $F(i)/V(a) =$ set of neighbors for node- i/a
 - ▶ Messages: $\mu_{i \rightarrow a}^{(t)}(x_i)$ and $\hat{\mu}_{a \rightarrow i}^{(t)}(x_i)$
- ▶ variable- i marginal $\mu_i^{(t+1)}(x)$

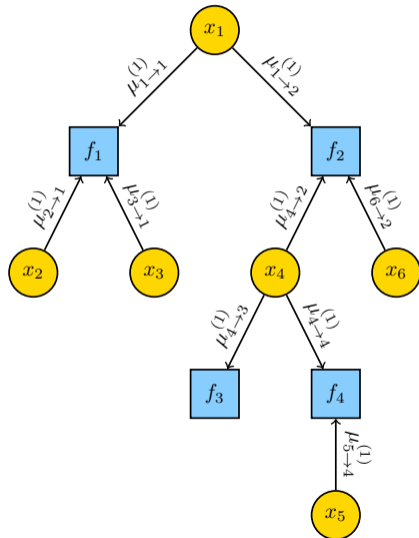


$$\mu_i^{(t+1)}(x_i) = \prod_{b \in F(i)} \hat{\mu}_{b \rightarrow i}^{(t)}(x_i)$$

Marginalization via Belief Propagation: Example

iteration 1: variable to factor

$$\mu_{i \rightarrow a}^{(1)}(x_i) = 1$$



Marginalization via Belief Propagation: Example

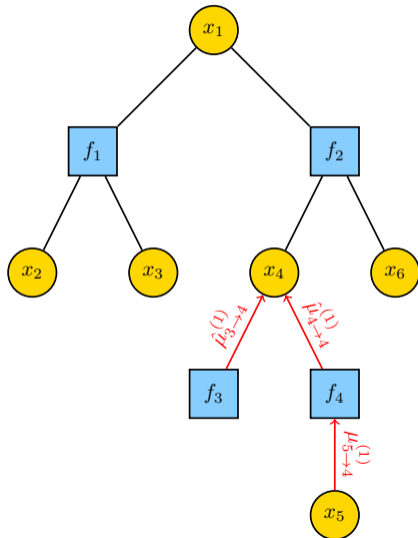
iteration 1: variable to factor

$$\mu_{i \rightarrow a}^{(1)}(x_i) = 1$$

iteration 1: factor to variable

$$\begin{aligned}\hat{\mu}_{4 \rightarrow 4}^{(1)}(x_4) &= \sum_{x_5} f_4(x_4, x_5) \mu_{5 \rightarrow 4}^{(1)}(x_5) \\ &= \sum_{x_5} f_4(x_4, x_5)\end{aligned}$$

$$\hat{\mu}_{3 \rightarrow 4}^{(1)}(x_4) = f_3(x_4)$$



Marginalization via Belief Propagation: Example

iteration 1: factor to variable

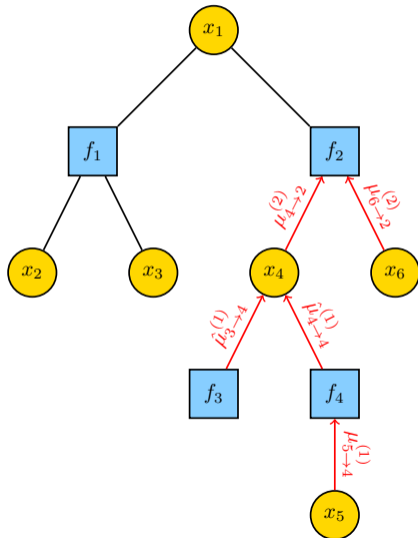
$$\begin{aligned}\hat{\mu}_{4 \rightarrow 4}^{(1)}(x_4) &= \sum_{x_5} f_4(x_4, x_5) \mu_{5 \rightarrow 4}^{(1)}(x_5) \\ &= \sum_{x_5} f_4(x_4, x_5)\end{aligned}$$

$$\hat{\mu}_{3 \rightarrow 4}^{(1)}(x_4) = f_3(x_4)$$

iteration 2: variable to factor

$$\begin{aligned}\mu_{4 \rightarrow 2}^{(2)}(x_4) &= \hat{\mu}_{4 \rightarrow 4}^{(1)}(x_4) \hat{\mu}_{3 \rightarrow 4}^{(1)}(x_4) \\ &= f_3(x_4) \sum_{x_5} f_4(x_4, x_5)\end{aligned}$$

$$\mu_{6 \rightarrow 2}^{(2)}(x_6) = 1$$



Marginalization via Belief Propagation: Example

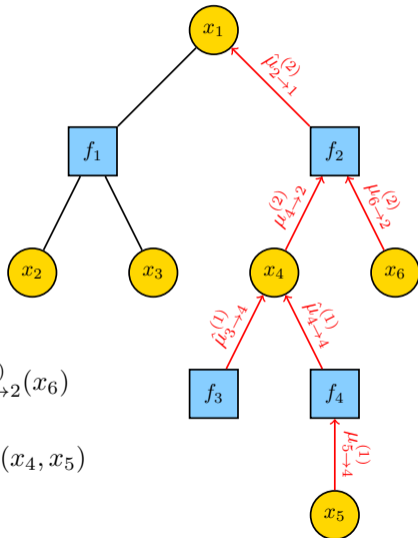
iteration 2: variable to factor

$$\begin{aligned}\mu_{4 \rightarrow 2}^{(2)}(x_4) &= \hat{\mu}_{4 \rightarrow 4}^{(1)}(x_4) \hat{\mu}_{3 \rightarrow 4}^{(1)}(x_4) \\ &= f_3(x_4) \sum_{x_5} f_4(x_4, x_5)\end{aligned}$$

$$\mu_{6 \rightarrow 2}^{(2)}(x_6) = 1$$

iteration 2: factor to variable

$$\begin{aligned}\hat{\mu}_{2 \rightarrow 1}^{(2)}(x_1) &= \sum_{x_4, x_6} f_2(x_1, x_4, x_6) \mu_{4 \rightarrow 2}^{(2)}(x_4) \mu_{6 \rightarrow 2}^{(2)}(x_6) \\ &= \sum_{x_4, x_6} f_2(x_1, x_4, x_6) f_3(x_4) \sum_{x_5} f_4(x_4, x_5) \\ &= f_2''(x_1)\end{aligned}$$

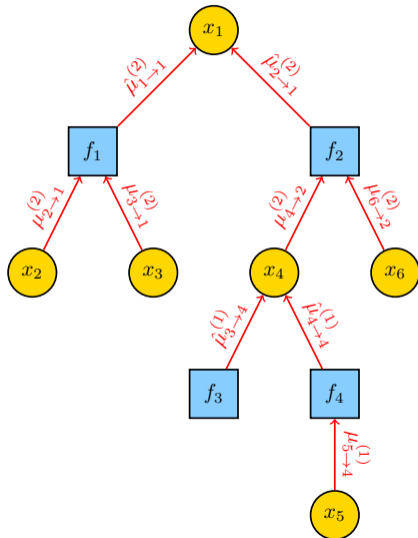


Marginalization via Belief Propagation: Example

iteration 2: variable marginal

$$\begin{aligned}\mu_1^{(3)}(x_1) &= \hat{\mu}_{1 \rightarrow 1}^{(2)}(x_1) \hat{\mu}_{2 \rightarrow 1}^{(2)}(x_1) \\ &= f_1''(x_1) f_2''(x_2)\end{aligned}$$

Same answer as peeling but from a distributed parallel algorithm

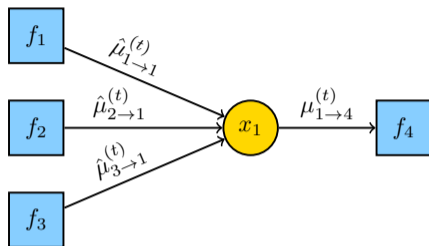


Log Likelihood-Ratio Messages for Binary Variables

- ▶ Normalized binary messages given by scalar: $\mu(1) = 1 - \mu(0)$
 - ▶ One can also use the likelihood ratio (LR): $\frac{\mu(0)}{\mu(1)}$
 - ▶ Or the log likelihood-ratio (LLR): $L = \ln \frac{\mu(0)}{\mu(1)}$
- ▶ For inference, LLR messages contain all the information:

$$L_{i \rightarrow a}^{(t)} = \ln \frac{\mu_{i \rightarrow a}^{(t)}(0)}{\mu_{i \rightarrow a}^{(t)}(1)} \qquad \hat{L}_{a \rightarrow i}^{(t)} = \ln \frac{\hat{\mu}_{a \rightarrow i}^{(t)}(0)}{\hat{\mu}_{a \rightarrow i}^{(t)}(1)}$$

VN Update for Binary Variables in LLR Domain



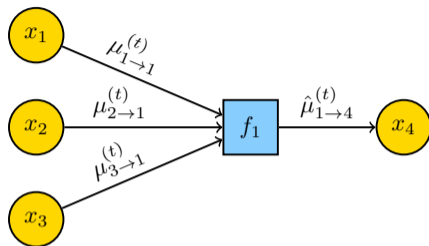
- ▶ Recall that the VN message-passing update is:

$$\mu_{i \rightarrow a}^{(t+1)}(x_i) = \prod_{b \in F(i) \setminus a} \hat{\mu}_{b \rightarrow i}^{(t)}(x_i)$$

- ▶ In the LLR domain, this simplifies to

$$L_{i \rightarrow a}^{(t+1)} = \ln \frac{\mu_{i \rightarrow a}^{(t+1)}(0)}{\mu_{i \rightarrow a}^{(t+1)}(1)} = \ln \frac{\prod_{b \in F(i) \setminus a} \hat{\mu}_{b \rightarrow i}^{(t)}(0)}{\prod_{b \in F(i) \setminus a} \hat{\mu}_{b \rightarrow i}^{(t)}(1)} = \sum_{b \in F(i) \setminus a} \hat{L}_{b \rightarrow i}^{(t)}$$

FN Update for Binary Variables in LLR Domain



- ▶ Recall that the FN message-passing update is:

$$\hat{\mu}_{a \rightarrow i}^{(t)}(x_i) = \sum_{x_{V(a)} \setminus x_i} f_1(x_{V(a)}) \prod_{j \in V(a) \setminus i} \mu_{j \rightarrow a}^{(t)}(x_j)$$

- ▶ In the LLR domain, this gives

$$\hat{L}_{a \rightarrow i}^{(t)} = \ln \frac{\hat{\mu}_{a \rightarrow i}^{(t)}(0)}{\hat{\mu}_{a \rightarrow i}^{(t)}(1)} = \ln \frac{\sum_{x_{V(a)}: x_i=0} f_1(x_{V(a)}) \prod_{j \in V(a) \setminus i} \mu_{j \rightarrow a}^{(t)}(x_j)}{\sum_{x_{V(a)}: x_i=1} f_1(x_{V(a)}) \prod_{j \in V(a) \setminus i} \mu_{j \rightarrow a}^{(t)}(x_j)}$$

FN Update for Even-Parity Constraint in LLR Domain

- ▶ The result of the even-parity constraint homework problem implies:

$$\begin{aligned}\hat{L}_{1 \rightarrow d}^{(t)} &= \ln \frac{\prod_{j=1}^{d-1} \left(\mu_{j \rightarrow 1}^{(t)}(0) + \mu_{j \rightarrow 1}^{(t)}(1) \right) + \prod_{j=1}^{d-1} \left(\mu_{j \rightarrow 1}^{(t)}(0) - \mu_{j \rightarrow 1}^{(t)}(1) \right)}{\prod_{j=1}^{d-1} \left(\mu_{j \rightarrow 1}^{(t)}(0) + \mu_{j \rightarrow 1}^{(t)}(1) \right) - \prod_{j=1}^{d-1} \left(\mu_{j \rightarrow 1}^{(t)}(0) - \mu_{j \rightarrow 1}^{(t)}(1) \right)} \\ &= \ln \frac{1 + \prod_{j=1}^{d-1} \frac{\mu_{j \rightarrow 1}^{(t)}(0) - \mu_{j \rightarrow 1}^{(t)}(1)}{\mu_{j \rightarrow 1}^{(t)}(0) + \mu_{j \rightarrow 1}^{(t)}(1)}}{1 - \prod_{j=1}^{d-1} \frac{\mu_{j \rightarrow 1}^{(t)}(0) - \mu_{j \rightarrow 1}^{(t)}(1)}{\mu_{j \rightarrow 1}^{(t)}(0) + \mu_{j \rightarrow 1}^{(t)}(1)}} \\ &= \ln \frac{1 + \prod_{j=1}^{d-1} \tanh \left(\frac{1}{2} L_{j \rightarrow 1}^{(t)} \right)}{1 - \prod_{j=1}^{d-1} \tanh \left(\frac{1}{2} L_{j \rightarrow 1}^{(t)} \right)} \quad \left(\tanh \left(\frac{1}{2} \ln \frac{a}{b} \right) = \frac{a-b}{a+b} \right) \\ &= 2 \tanh^{-1} \left(\prod_{j=1}^{d-1} \tanh \left(\frac{1}{2} L_{j \rightarrow 1}^{(t)} \right) \right) \quad \left(2 \tanh^{-1}(z) = \ln \frac{1+z}{1-z} \right)\end{aligned}$$

Outline

Low-Density Parity-Check Codes

Factor Graphs

Message Passing

LDPC Ensembles

Overview of Code Ensembles

- ▶ (j, k) -regular ensemble of parity-check matrices
 - ▶ Every column has j ones and every row has k ones
 - ▶ Length- n code has parity-check matrix with n columns and $m = nj/k$ rows
 - ▶ Variety of similar constructions related to random biregular bipartite graphs
 - ▶ Regular structure is nice but **performance can be improved by irregularity**

Overview of Code Ensembles

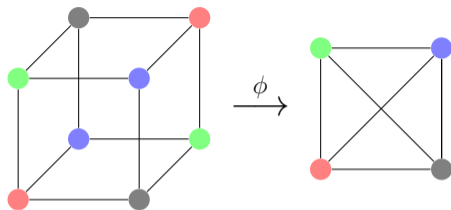
- ▶ (j, k) -regular ensemble of parity-check matrices
 - ▶ Every column has j ones and every row has k ones
 - ▶ Length- n code has parity-check matrix with n columns and $m = nj/k$ rows
 - ▶ Variety of similar constructions related to random biregular bipartite graphs
 - ▶ Regular structure is nice but **performance can be improved by irregularity**
- ▶ Standard (Λ, P) -irregular ensemble where $\Lambda(x) = \sum_i \Lambda_i x^i$ and $P(x) = \sum_i P_i x^i$
 - ▶ Code length $n = \Lambda(1)$ with $m = P(1)$ parity checks
 - ▶ # of columns with i ones is Λ_i and # of rows with i ones is P_i and
 - ▶ Configuration model where permutation used to match edge sockets from nodes
 - ▶ At moderate length, performance hurt by **variation in local neighborhood structure**

Overview of Code Ensembles

- ▶ (j, k) -regular ensemble of parity-check matrices
 - ▶ Every column has j ones and every row has k ones
 - ▶ Length- n code has parity-check matrix with n columns and $m = nj/k$ rows
 - ▶ Variety of similar constructions related to random biregular bipartite graphs
 - ▶ Regular structure is nice but **performance can be improved by irregularity**
- ▶ Standard (Λ, P) -irregular ensemble where $\Lambda(x) = \sum_i \Lambda_i x^i$ and $P(x) = \sum_i P_i x^i$
 - ▶ Code length $n = \Lambda(1)$ with $m = P(1)$ parity checks
 - ▶ # of columns with i ones is Λ_i and # of rows with i ones is P_i and
 - ▶ Configuration model where permutation used to match edge sockets from nodes
 - ▶ At moderate length, performance hurt by **variation in local neighborhood structure**
- ▶ Protograph ensemble uses small matrix H^B to define larger parity-check matrix
 - ▶ Allows **controlled irregularity to obtain very good performance**
 - ▶ Described in the next few slides

Protograph LDPC Code Ensemble (1)

In topology, covering maps are used to describe large objects that are locally indistinguishable from smaller objects. If the large object can be mapped down to the smaller object in a way that preserves all local properties, then the mapping is called a covering map. This idea is used to construct irregular LDPC codes with deterministic local structure.



The figure shows an example covering map, ϕ , that maps each vertex of the cube to the unique vertex in K_4 (i.e., complete graph on 4 vertices) that has the same color.

Definition

Let B be a **base graph** with no multiple edges and G be an arbitrary graph. A function, $\phi : G \rightarrow B$, that maps the vertices of G to the vertices of B is a **covering map** if, for every $x \in G$, the neighbors of x are bijectively mapped to the neighbors of $\phi(x)$ in B . We say that G is a **graph cover** of B .

Protograph LDPC Code Ensemble (2)

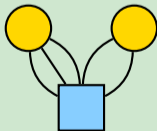
Simple code designs often use protographs that have multiple edges (i.e., are multigraphs). Though the previous definition does not generalize, one can still use them as base graphs.

Definition

Let B be a bipartite multigraph with $(N - K) \times N$ adjacency matrix H^B . Let H^G be an $m(N - K) \times mN$ matrix formed by replacing the (i, j) entry of H^B with the sum of $H_{i,j}^B$ $m \times m$ permutation matrices. Then, G is an m -lift of B . If B has no multiple edges, then G is called a graph cover of B . Using

Example

For the base multigraph with $H^B = \begin{bmatrix} 3 & 2 \end{bmatrix}$, here is a 3-lift with no multiple edges:



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

Protograph LDPC Code Ensemble (3)

Definition

For a bipartite multigraph B , the **protograph code ensemble** $\text{PROTO}(B, m)$ is defined by choosing the Tanner graph G to be an m -lift of B where each permutation matrix is chosen uniformly at random. The resulting code is defined by the parity-check matrix H^G .

Example

For the protograph ensemble $\text{PROTO}(B, m)$, each row and column in the parity-check matrix H^B of the base graph B define a **node type**. The local neighborhood of each node in the lifted graph is deterministic and depends only on its node type (i.e., its associated node in the base graph). Likewise, each non-zero entry in the parity-check matrix H^B of the base graph defines an **edge type**. The local neighborhood of each edge in the lifted graph is deterministic and depends only on its edge type.