

Introduction to Factor Graphs and Belief Propagation

Henry D. Pfister

Graphical Models and Inference
Duke University
Spring 2026

Outline

Factor Graphs

Message Passing

Applications of Factor Graphs

Outline

Factor Graphs

Message Passing

Applications of Factor Graphs

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:

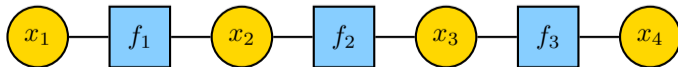
$$\begin{aligned} P(x_1, x_2, x_3, x_4) &\triangleq \mathbb{P}(X_1 = x_1, X_2 = x_2, \dots, X_4 = x_4 | Y = y) \\ &\propto f(x_1, x_2, x_3, x_4) \\ &\triangleq f_1(x_1, x_2) f_2(x_2, x_3) f_3(x_3, x_4) \end{aligned}$$

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 from A to B that avoids S), then we have $X_A \perp\!\!\!\perp X_B \mid X_S$

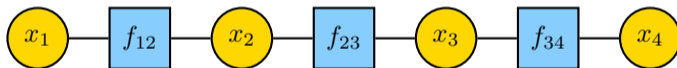
$$P(x_A, x_B | x_S) = P(x_A | x_S)P(x_B | x_S)$$

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 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: $A = \{x_1, x_2\}$, $B = \{x_4\}$, $S = \{x_3\}$

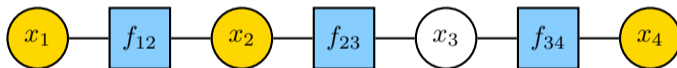


Conditional Independence for Factor Graphs

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



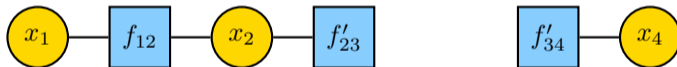
- ▶ Sketch of Proof:
 - ▶ Fixing $X_S = x_S$ separates the FG into disjoint components

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 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
 - ▶ 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:

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

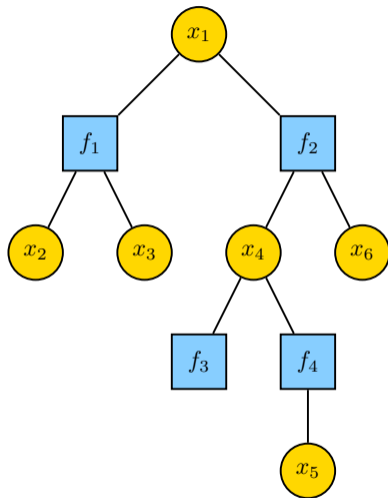
The Importance of Factorization (2)

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

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

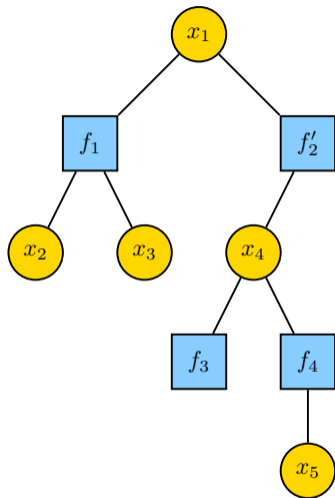
This implementation requires roughly $2|\mathcal{X}|^3 + O(|\mathcal{X}|^2)$ operations

The Factor Graph and Leaf Removal



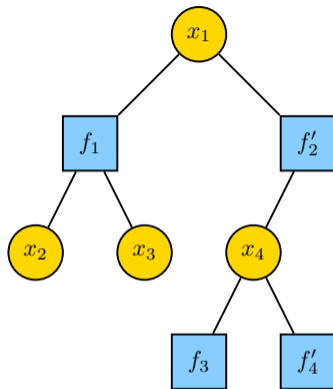
$$g_1(x_1) = \sum_{x_2^5} f_1(x_1, x_2, x_3) f_3(x_4) f_4(x_4, x_5) \sum_{x_6} f_2(x_1, x_4, x_6)$$

The Factor Graph and Leaf Removal



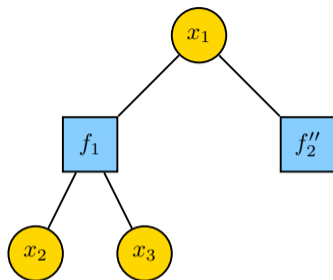
$$g_1(x_1) = \sum_{x_2} f_1(x_1, x_2, x_3) f_3(x_4) \left[\sum_{x_5} f_4(x_4, x_5) \right] f_2'(x_1, x_4)$$

The Factor Graph and Leaf Removal



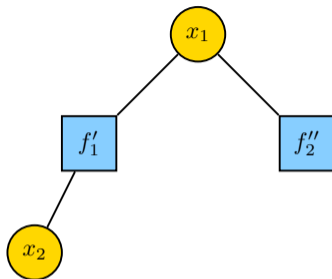
$$g_1(x_1) = \sum_{x_2} f_1(x_1, x_2, x_3) \left[\sum_{x_4} f_3(x_4) f'_4(x_4) f'_2(x_1, x_4) \right]$$

The Factor Graph and Leaf Removal



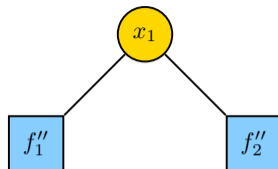
$$g_1(x_1) = \sum_{x_2} \left[\sum_{x_3} f_1(x_1, x_2, x_3) \right] f_2''(x_1)$$

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

Outline

Factor Graphs

Message Passing

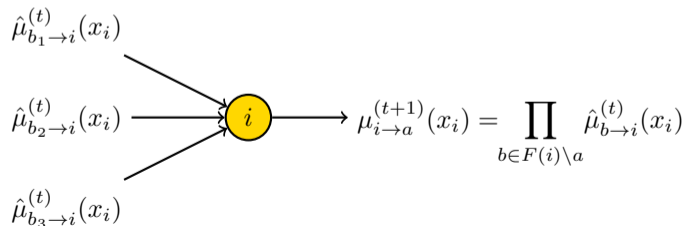
Applications of Factor Graphs

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 $\hat{\mu}_{a \rightarrow i}^{(t)}(x_i)$

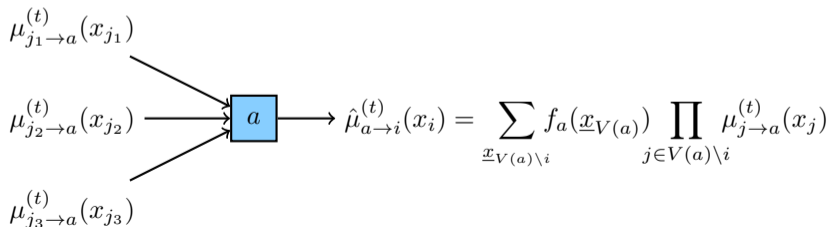
Marginalization via Belief Propagation

- ▶ 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

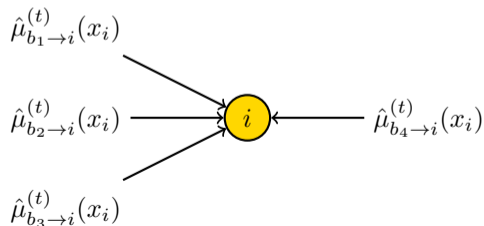


The diagram shows a central blue square node labeled 'a'. Three arrows point towards it from the left, labeled $\mu_{j_1 \rightarrow a}^{(t)}(x_{j_1})$, $\mu_{j_2 \rightarrow a}^{(t)}(x_{j_2})$, and $\mu_{j_3 \rightarrow a}^{(t)}(x_{j_3})$. An arrow points from node 'a' to the right, leading to the equation:

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

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 $\mu_i^{(t+1)}(x)$

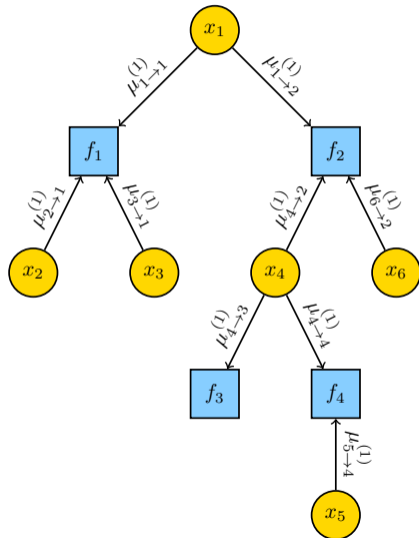


$$\mu_i^{(t+1)}(x_i) = \prod_{b \in F(i)} \hat{\mu}_{b \rightarrow i}^{(t)}(x_i)$$

Marginalization via Belief Propagation: Example

iteration 1: variable to factor

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



Marginalization via Belief Propagation: Example

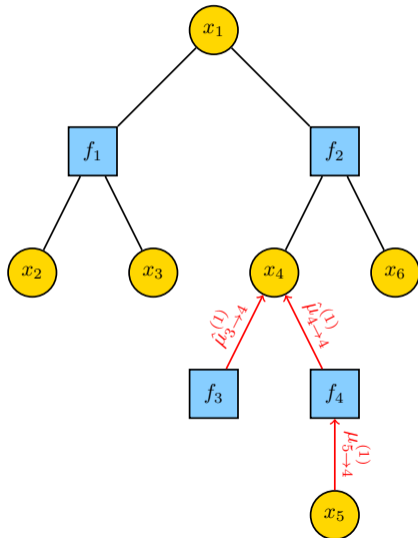
iteration 1: variable to factor

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

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

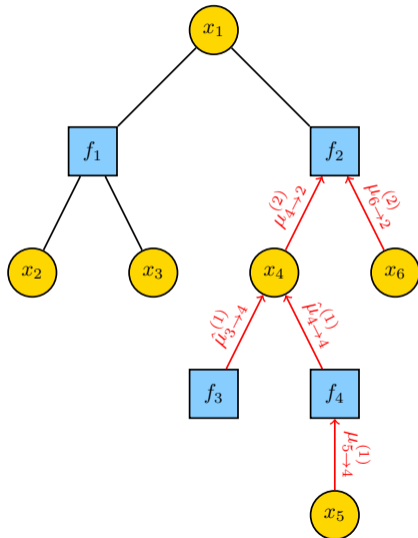
$$\begin{aligned}\hat{\mu}_{4 \rightarrow 4}^{(1)}(x_4) &= \sum_{x_5} f_4(x_4, x_5) \mu_{5 \rightarrow 4}^{(1)}(x_5) \\ &= \sum_{x_5} f_4(x_4, x_5)\end{aligned}$$

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

iteration 2: variable to factor

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

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

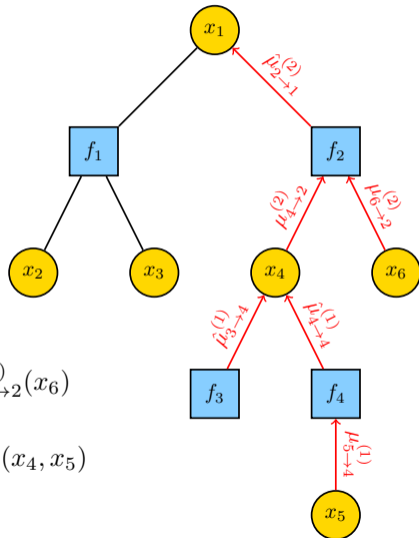


Marginalization via Belief Propagation: Example

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

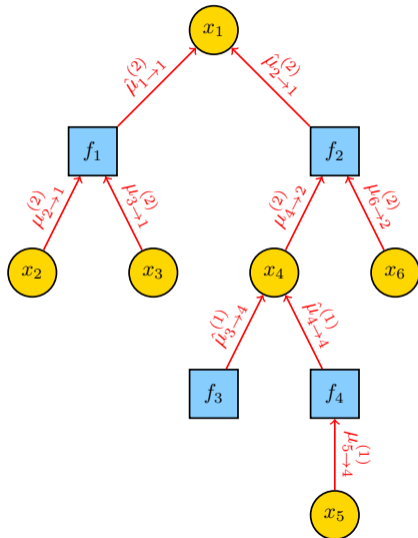
$$\begin{aligned}\hat{\mu}_{2 \rightarrow 1}^{(2)}(x_1) &= \sum_{x_4, x_6} f_2(x_1, x_4, x_6) \mu_{4 \rightarrow 2}^{(2)}(x_4) \mu_{6 \rightarrow 2}^{(2)}(x_6) \\ &= \sum_{x_4, x_6} f_2(x_1, x_4, x_6) f_3(x_4) \sum_{x_5} f_4(x_4, x_5) \\ &= f_2''(x_1)\end{aligned}$$

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

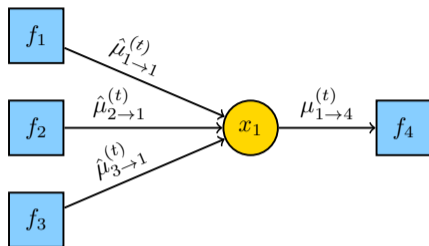


Log Likelihood-Ratio Messages for Binary Variables

- ▶ 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



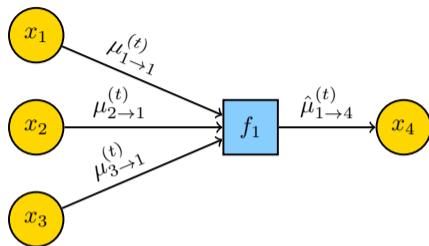
- ▶ Recall that the VN message-passing update is:

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

- ▶ 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

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

Outline

Factor Graphs

Message Passing

Applications of Factor Graphs

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

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

Solving Sudoku with a Factor Graph

- ▶ Consider any **constraint satisfaction problem** with observed entries
 - ▶ One can write $f(\underline{x})$ as the **product of indicator functions**
 - ▶ Some factors force \underline{x} to be **valid** (i.e., satisfy constraints)
 - ▶ Other factors force \underline{x} to be **compatible** with observed values
 - ▶ Summing over \underline{x} counts the **#** of **valid compatible** sequences

Solving Sudoku with a Factor Graph

- ▶ Consider any **constraint satisfaction problem** with observed entries
 - ▶ One can write $f(\underline{x})$ as the **product of indicator functions**
 - ▶ Some factors force \underline{x} to be **valid** (i.e., satisfy constraints)
 - ▶ Other factors force \underline{x} to be **compatible** with observed values
 - ▶ Summing over \underline{x} counts the $\#$ of **valid compatible** sequences
- ▶ Low-complexity **peeling solution**
 - ▶ Set elements of \underline{x} one at a time
 - ▶ Each step looks for $i \in [n]$ and $x' \in \mathcal{X}$ such that:
 - ▶ For currently set variables, $f(\underline{x}) = 0$ for all $x_i \in \mathcal{X} \setminus x'$
 - ▶ Sudoku's unique solution implies that $x_i = x'$ **correct**
 - ▶ Fix $x_i = x'$ and repeat until all values fixed

Boolean Satisfiability: K-SAT

- ▶ One instance of 3-SAT is given, for example, by

$$f(\underline{x}) = (\bar{x}_1 \vee \bar{x}_3 \vee x_7) \wedge (x_1 \vee \bar{x}_2 \vee x_5) \wedge (x_2 \vee \bar{x}_4 \vee x_6).$$

- ▶ In the FG, clause $a \in [m]$ is enforced by the function f_a

Boolean Satisfiability: K-SAT

- ▶ One instance of 3-SAT is given, for example, by

$$f(\underline{x}) = (\bar{x}_1 \vee \bar{x}_3 \vee x_7) \wedge (x_1 \vee \bar{x}_2 \vee x_5) \wedge (x_2 \vee \bar{x}_4 \vee x_6).$$

- ▶ In the FG, clause $a \in [m]$ is enforced by the function f_a
- ▶ Marginalization allows uniform sampling from **valid** set
 - ▶ For $i = 1, 2, \dots, n$, fix x_j for $j < i$ and compute marginal

$$g_i(x_i) = \frac{1}{Z_i} \sum_{x_{i+1}, \dots, x_n} f(\underline{x}) = \mathbb{P}(X_i = x_i | X_1 = x_1, \dots, X_{i-1} = x_{i-1})$$

- ▶ Then, sample $x_i \sim g_i(\cdot)$ and repeat

Boolean Satisfiability: K-SAT

- ▶ One instance of 3-SAT is given, for example, by

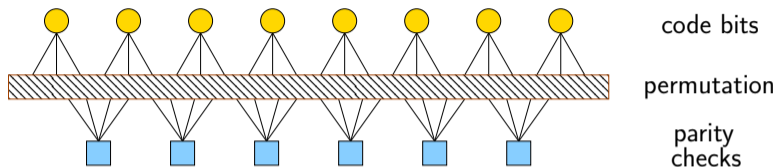
$$f(\underline{x}) = (\bar{x}_1 \vee \bar{x}_3 \vee x_7) \wedge (x_1 \vee \bar{x}_2 \vee x_5) \wedge (x_2 \vee \bar{x}_4 \vee x_6).$$

- ▶ In the FG, clause $a \in [m]$ is enforced by the function f_a
- ▶ Marginalization allows uniform sampling from **valid** set
 - ▶ For $i = 1, 2, \dots, n$, fix x_j for $j < i$ and compute marginal

$$g_i(x_i) = \frac{1}{Z_i} \sum_{x_{i+1}, \dots, x_n} f(\underline{x}) = \mathbb{P}(X_i = x_i | X_1 = x_1, \dots, X_{i-1} = x_{i-1})$$

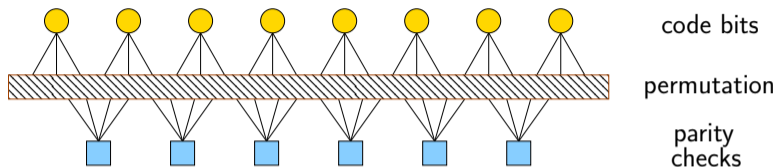
- ▶ Then, sample $x_i \sim g_i(\cdot)$ and repeat
- ▶ This algorithm has **low complexity** if factor graph forms a tree
 - ▶ If not a tree, use approximate marginal from belief propagation
 - ▶ This is related to **BP-guided decimation** [MM09]

Low-Density Parity-Check (LDPC) Codes



- ▶ Linear codes defined by $\underline{x}H^T = \underline{0}$ for all c.w. $\underline{x} \in \mathcal{C} \subset \{0, 1\}^n$
 - ▶ H is an $m \times n$ sparse parity-check matrix for the code
 - ▶ Code bits and parity checks associated with cols/rows of H

Low-Density Parity-Check (LDPC) Codes



- ▶ Linear codes defined by $\underline{x}H^T = \underline{0}$ for all c.w. $\underline{x} \in \mathcal{C} \subset \{0, 1\}^n$
 - ▶ H is an $m \times n$ sparse parity-check matrix for the code
 - ▶ Code bits and parity checks associated with cols/rows of H
- ▶ Factor graph: H is the biadjacency matrix for variable/factor nodes
 - ▶ Ensemble defined by configuration model for random graphs
 - ▶ Checks define factors: $f_{\text{even}}(x_1^d) = \mathbb{I}(x_1 \oplus \dots \oplus x_d = 0)$
 - ▶ Let $\underline{x}_{\partial a}$ be the subvector of variables in the a -th check and

$$f(x_1, \dots, x_n) = \left(\prod_{a=1}^m f_{\text{even}}(\underline{x}_{\partial a}) \right) \left(\prod_{i=1}^n P_{Y|X}(y_i|x_i) \right)$$

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

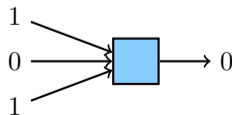
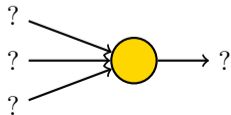
Belief networks are directed acyclic graphs in which the nodes represent propositions (or variables), the arcs signify direct dependencies between the linked propositions, and the strengths of these dependencies are quantified by conditional probabilities. A network of this sort can be used to represent 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.

Simple Message-Passing Decoding for the BEC

- ▶ Constraint nodes define the valid patterns
 - ▶ **Circles** represent a single value shared by factors
 - ▶ **Squares** assert attached variables sum to 0 mod 2
- ▶ Iterative decoding on the binary erasure channel (BEC)
 - ▶ Messages passed in phases: bit-to-check and check-to-bit
 - ▶ Each **output message depends on other input messages**
 - ▶ Each message is **either the correct value or an erasure**

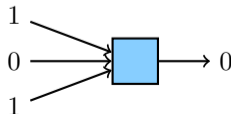
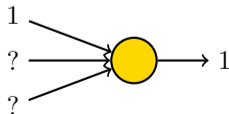
Simple Message-Passing Decoding for the BEC

- ▶ Constraint nodes define the valid patterns
 - ▶ **Circles** represent a single value shared by factors
 - ▶ **Squares** assert attached variables sum to 0 mod 2
- ▶ Iterative decoding on the binary erasure channel (BEC)
 - ▶ Messages passed in phases: bit-to-check and check-to-bit
 - ▶ Each **output message depends on other input messages**
 - ▶ Each message is **either the correct value or an erasure**
- ▶ Message passing rules for the BEC
 - ▶ Bits pass an erasure only if all other inputs are erased
 - ▶ Checks pass the correct value only if all other inputs are correct



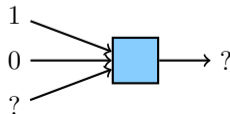
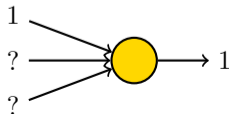
Simple Message-Passing Decoding for the BEC

- ▶ Constraint nodes define the valid patterns
 - ▶ **Circles** represent a single value shared by factors
 - ▶ **Squares** assert attached variables sum to 0 mod 2
- ▶ Iterative decoding on the binary erasure channel (BEC)
 - ▶ Messages passed in phases: bit-to-check and check-to-bit
 - ▶ Each **output message depends on other input messages**
 - ▶ Each message is **either the correct value or an erasure**
- ▶ Message passing rules for the BEC
 - ▶ Bits pass an erasure only if all other inputs are erased
 - ▶ Checks pass the correct value only if all other inputs are correct

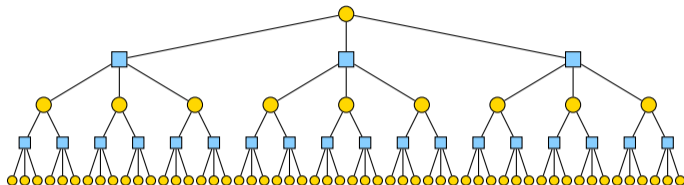


Simple Message-Passing Decoding for the BEC

- ▶ Constraint nodes define the valid patterns
 - ▶ **Circles** represent a single value shared by factors
 - ▶ **Squares** assert attached variables sum to 0 mod 2
- ▶ Iterative decoding on the binary erasure channel (BEC)
 - ▶ Messages passed in phases: bit-to-check and check-to-bit
 - ▶ Each **output message depends on other input messages**
 - ▶ Each message is **either the correct value or an erasure**
- ▶ Message passing rules for the BEC
 - ▶ Bits pass an erasure only if all other inputs are erased
 - ▶ Checks pass the correct value only if all other inputs are correct



Computation Graph and Density Evolution



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

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

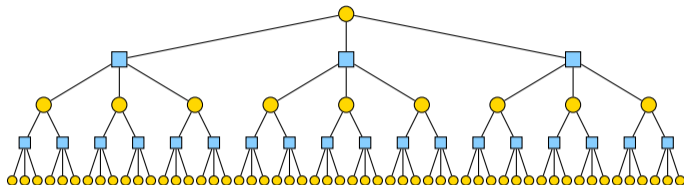
$$x_2 = \varepsilon y_1^2$$

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

$$x_1 = \varepsilon$$

- ▶ Computation graph for a (3,4)-regular LDPC code
 - ▶ Illustrates decoding from the **perspective of a single bit-node**
 - ▶ For long random LDPC codes, the graph is typically a tree
 - ▶ Allows density evolution to **track message erasure probability**

Computation Graph and Density Evolution



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

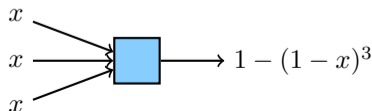
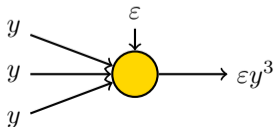
$$y_2 = 1 - (1 - x_2)^3$$

$$x_2 = \varepsilon y_1^2$$

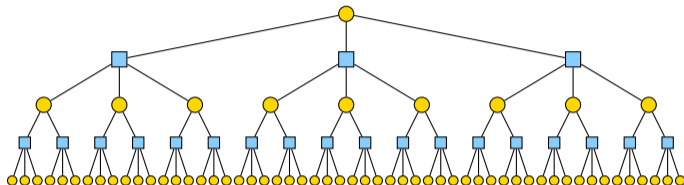
$$y_1 = 1 - (1 - x_1)^3$$

$$x_1 = \varepsilon$$

- ▶ Computation graph for a (3,4)-regular LDPC code
 - ▶ Illustrates decoding from the **perspective of a single bit-node**
 - ▶ For long random LDPC codes, the graph is typically a tree
 - ▶ Allows density evolution to **track message erasure probability**
 - ▶ If x/y are erasure prob. of bit/check output messages, then



Computation Graph and Density Evolution



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

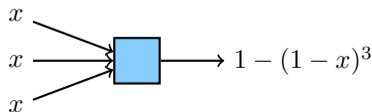
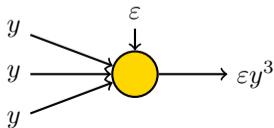
$$y_2 = 1 - (1 - x_2)^3$$

$$x_2 = \varepsilon y_1^2$$

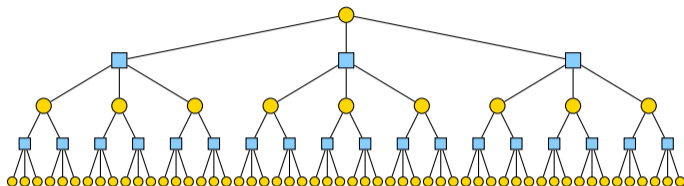
$$y_1 = 0.936$$

$$x_1 = 0.600$$

- ▶ Computation graph for a (3,4)-regular LDPC code
 - ▶ Illustrates decoding from the **perspective of a single bit-node**
 - ▶ For long random LDPC codes, the graph is typically a tree
 - ▶ Allows density evolution to **track message erasure probability**
 - ▶ If x/y are erasure prob. of bit/check output messages, then



Computation Graph and Density Evolution



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

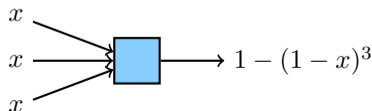
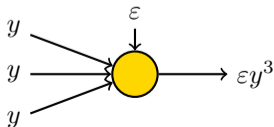
$$y_2 = 1 - (1 - x_2)^3$$

$$x_2 = 0.526$$

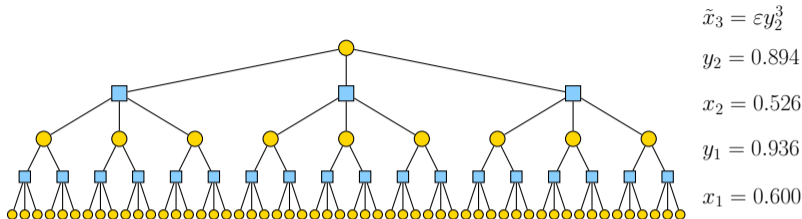
$$y_1 = 0.936$$

$$x_1 = 0.600$$

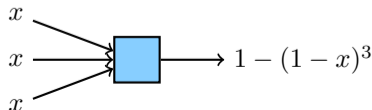
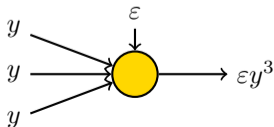
- ▶ Computation graph for a (3,4)-regular LDPC code
 - ▶ Illustrates decoding from the **perspective of a single bit-node**
 - ▶ For long random LDPC codes, the graph is typically a tree
 - ▶ Allows density evolution to **track message erasure probability**
 - ▶ If x/y are erasure prob. of bit/check output messages, then



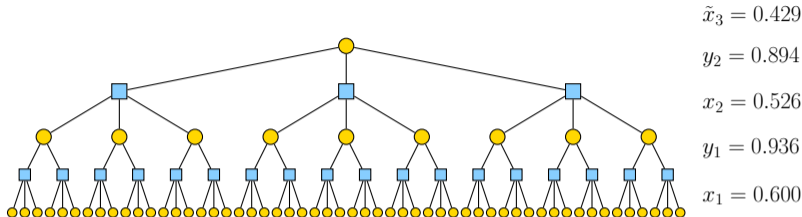
Computation Graph and Density Evolution



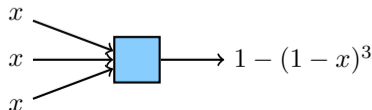
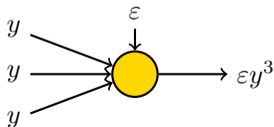
- ▶ Computation graph for a (3,4)-regular LDPC code
 - ▶ Illustrates decoding from the **perspective of a single bit-node**
 - ▶ For long random LDPC codes, the graph is typically a tree
 - ▶ Allows density evolution to **track message erasure probability**
 - ▶ If x/y are erasure prob. of bit/check output messages, then



Computation Graph and Density Evolution

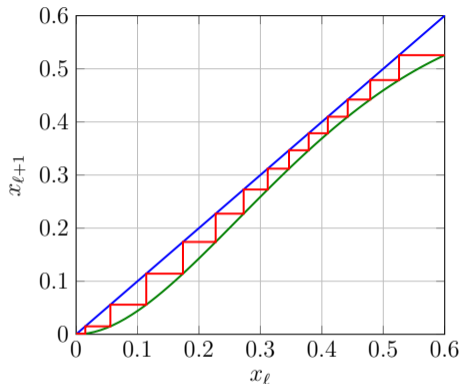


- ▶ Computation graph for a (3,4)-regular LDPC code
 - ▶ Illustrates decoding from the **perspective of a single bit-node**
 - ▶ For long random LDPC codes, the graph is typically a tree
 - ▶ Allows density evolution to **track message erasure probability**
 - ▶ If x/y are erasure prob. of bit/check output messages, then



Density Evolution (DE) for LDPC Codes

(3,4) LDPC Code with $\varepsilon = 0.6$



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

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

Decoding Thresholds:

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

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

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

- ▶ Binary erasure channel (BEC) with erasure prob. ε
- ▶ DE tracks bit-to-check msg erasure rate x_{ℓ} after ℓ iterations
- ▶ Defines **noise threshold** ε^{BP} for the large system limit
 - ▶ Easily computed numerically for given code ensemble

References I

- [Gal63] Robert G. Gallager. *Low-Density Parity-Check Codes*.
The M.I.T. Press, Cambridge, MA, USA, 1963.
- [KFL01] Frank R. Kschischang, Brendan J. Frey, and Hans-Andrea Loeliger.
Factor graphs and the sum-product algorithm.
IEEE Trans. Inform. Theory, 47(2):498–519, Feb. 2001.
- [LMSS01] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman.
Efficient erasure correcting codes.
IEEE Trans. Inform. Theory, 47(2):569–584, Feb. 2001.
- [Mac99] David J. C. MacKay.
Good error-correcting codes based on very sparse matrices.
IEEE Trans. Inform. Theory, 45(2):399–431, March 1999.
- [MM09] M. Mezard and A. Montanari.
Information, Physics, and Computation.
Oxford University Press, New York, NY, 2009.
- [RSU01] Thomas J. Richardson, M. Amin Shokrollahi, and Rüdiger L. Urbanke.
Design of capacity-approaching irregular low-density parity-check codes.
IEEE Trans. Inform. Theory, 47(2):619–637, Feb. 2001.

References II

- [RU01] Thomas J. Richardson and Rüdiger L. Urbanke.
The capacity of low-density parity-check codes under message-passing decoding.
IEEE Trans. Inform. Theory, 47(2):599–618, Feb. 2001.
- [RU08] Thomas J. Richardson and Rüdiger L. Urbanke.
Modern Coding Theory.
Cambridge University Press, New York, NY, 2008.