



# Probabilistic Graphical Models in Computer Vision (IN2329)

Csaba Domokos

Summer Semester 2017



Sum-product algorithm

Max-sum algorithm

Loopy belief propagation

## 8. Belief Propagation



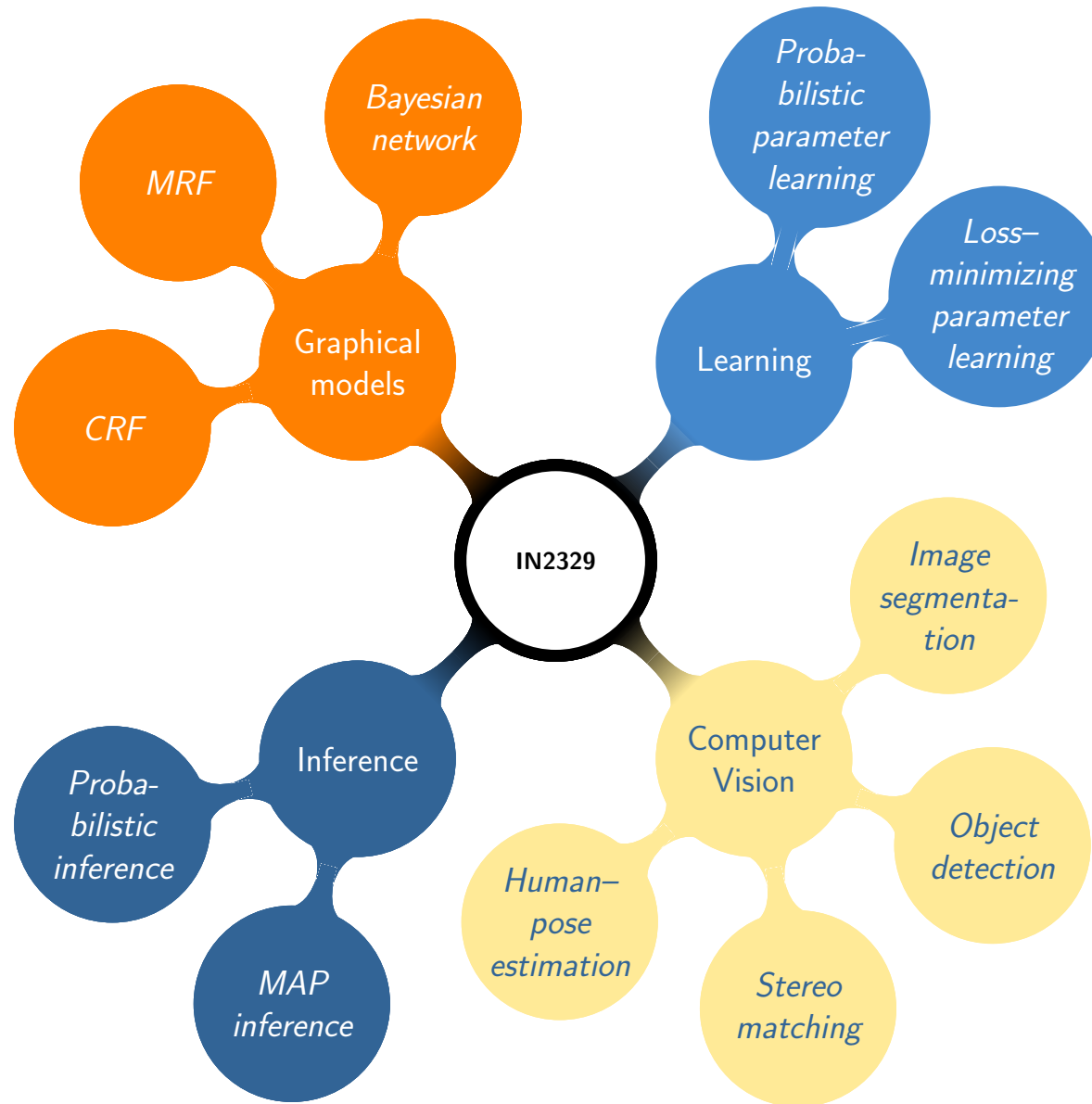
# Overview of the course \*



Sum-product algorithm

Max-sum algorithm

Loopy belief propagation





# Recall: Inference \*



Sum-product algorithm

Max-sum algorithm

Loopy belief propagation

**Inference** means the procedure to estimate the *probability distribution*, encoded by a *graphical model*, for a *given data* (or observation).

Assume we are given a factor graph  $G = (\mathcal{V}, \mathcal{E}', \mathcal{F})$  and the observation  $\mathbf{x}$ .

- **Maximum A Posteriori (MAP) inference:** find the *state*  $\mathbf{y}^* \in \mathcal{Y}$  of *maximum probability*,

$$\mathbf{y}^* \in \operatorname{argmax}_{\mathbf{y} \in \mathcal{Y}} p(\mathbf{y} \mid \mathbf{x}) = \operatorname{argmin}_{\mathbf{y} \in \mathcal{Y}} E(\mathbf{y}; \mathbf{x}) .$$

- **Probabilistic inference:** find the value of the *partition function*  $Z(\mathbf{x})$  and the *marginal distributions*  $\mu_F(\mathbf{y}_F)$  for each factor  $F \in \mathcal{F}$ ,

$$Z(\mathbf{x}) = \sum_{\mathbf{y} \in \mathcal{Y}} \exp(-E(\mathbf{y}; \mathbf{x})) ,$$

$$\mu_F(\mathbf{y}_F) = p(\mathbf{y}_F \mid \mathbf{x}) .$$

# Agenda for today's lecture \*

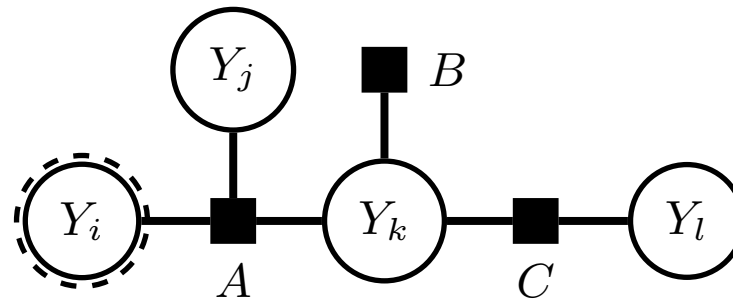


Sum-product algorithm

Max-sum algorithm

Loopy belief propagation

Today we are going to learn about **belief propagation** to perform **exact** inference on graphical models having **tree structure**.



*Reminder:* a tree is a connected and acyclic graph.

- Probabilistic inference: *Sum-product algorithm*
- MAP inference: *Max-sum algorithm*

We also extend belief propagation for **general** factor graphs, which results in an **approximate** inference.



Sum-product algorithm

Max-sum algorithm

Loopy belief propagation

# Sum-product algorithm

# Probabilistic inference on chains

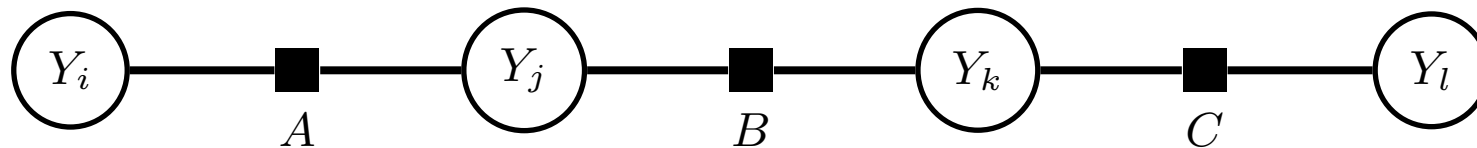


Sum-product algorithm

Max-sum algorithm

Loopy belief propagation

Assume that we are given the following factor graph and a corresponding energy function  $E(\mathbf{y})$ , where  $\mathcal{Y} = \mathcal{Y}_i \times \mathcal{Y}_j \times \mathcal{Y}_k \times \mathcal{Y}_l$ .



We want to compute  $p(\mathbf{y})$  for any  $\mathbf{y} \in \mathcal{Y}$  by making use of the factorization

$$p(\mathbf{y}) = \frac{1}{Z} \exp(-E(\mathbf{y})) = \frac{1}{Z} \exp(-E_A(y_i, y_j)) \exp(-E_B(y_j, y_k)) \exp(-E_C(y_k, y_l)).$$

**Problem:** we also need to calculate the *partition function*

$$Z = \sum_{\mathbf{y} \in \mathcal{Y}} \exp(-E(\mathbf{y})) = \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \sum_{y_k \in \mathcal{Y}_k} \sum_{y_l \in \mathcal{Y}_l} \exp(-E(y_i, y_j, y_k, y_l)),$$

which looks expensive (the sum has  $|\mathcal{Y}_i| \cdot |\mathcal{Y}_j| \cdot |\mathcal{Y}_k| \cdot |\mathcal{Y}_l|$  terms).

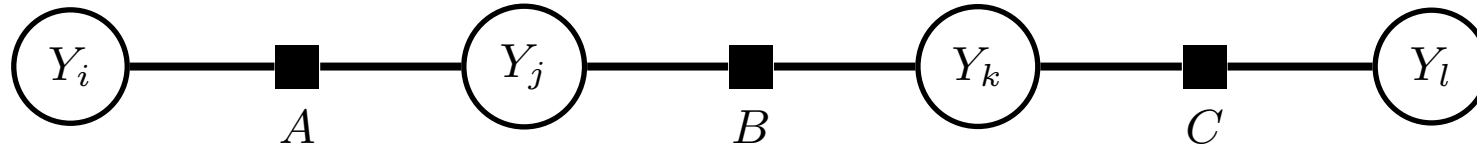
# Partition function



Sum-product algorithm

Max-sum algorithm

Loopy belief propagation



We can expand the *partition function* as

$$\begin{aligned} Z &= \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \sum_{y_k \in \mathcal{Y}_k} \sum_{y_l \in \mathcal{Y}_l} \exp(-E(y_i, y_j, y_k, y_l)) \\ &= \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \sum_{y_k \in \mathcal{Y}_k} \sum_{y_l \in \mathcal{Y}_l} \exp\left(-\left(E_A(y_i, y_j) + E_B(y_j, y_k) + E_C(y_k, y_l)\right)\right) \\ &= \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \sum_{y_k \in \mathcal{Y}_k} \sum_{y_l \in \mathcal{Y}_l} \exp(-E_A(y_i, y_j)) \exp(-E_B(y_j, y_k)) \exp(-E_C(y_k, y_l)) \\ &= \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \exp(-E_A(y_i, y_j)) \sum_{y_k \in \mathcal{Y}_k} \exp(-E_B(y_j, y_k)) \sum_{y_l \in \mathcal{Y}_l} \exp(-E_C(y_k, y_l)) . \end{aligned}$$



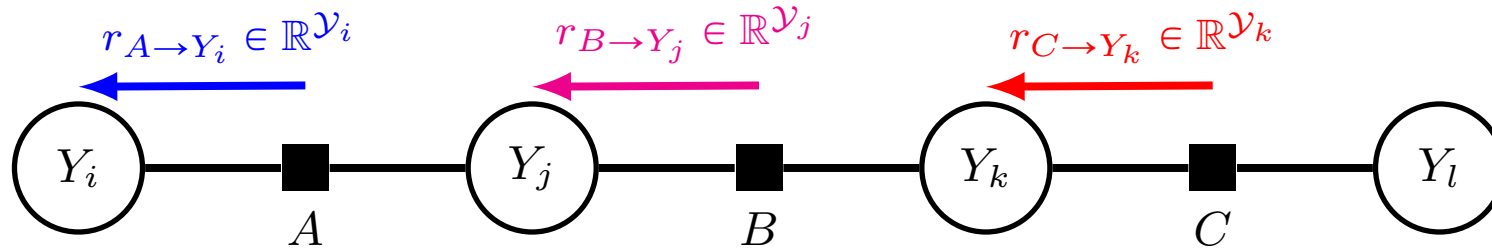
# Elimination



Sum-product algorithm

Max-sum algorithm

Loopy belief propagation



Note that we can successively *eliminate* variables, that is

$$\begin{aligned}
 Z &= \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \exp(-E_A(y_i, y_j)) \sum_{y_k \in \mathcal{Y}_k} \exp(-E_B(y_j, y_k)) \underbrace{\sum_{y_l \in \mathcal{Y}_l} \exp(-E_C(y_k, y_l))}_{r_{C \rightarrow Y_k}(y_k)} \\
 &= \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \exp(-E_A(y_i, y_j)) \underbrace{\sum_{y_k \in \mathcal{Y}_k} \exp(-E_B(y_j, y_k)) r_{C \rightarrow Y_k}(y_k)}_{r_{B \rightarrow Y_j}(y_j)} \\
 &= \sum_{y_i \in \mathcal{Y}_i} \underbrace{\sum_{y_j \in \mathcal{Y}_j} \exp(-E_A(y_i, y_j)) r_{B \rightarrow Y_j}(y_j)}_{r_{A \rightarrow Y_i}(y_i)} = \sum_{y_i \in \mathcal{Y}_i} r_{A \rightarrow Y_i}(y_i) .
 \end{aligned}$$

# Inference on trees

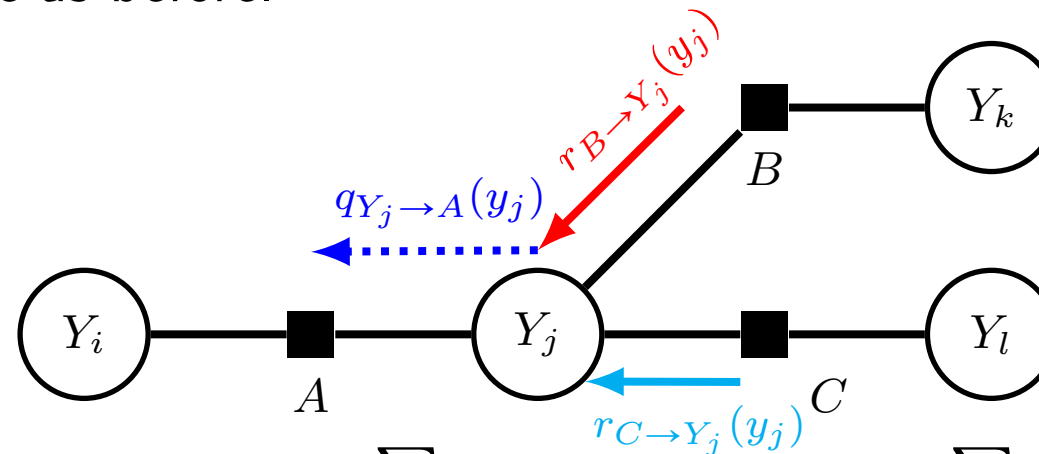


Sum-product algorithm

Max-sum algorithm

Loopy belief propagation

Now we are assuming a tree-structured factor graph and are applying the same elimination procedure as before.



$$\begin{aligned}
 Z &= \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \exp(-E_A(y_i, y_j)) \underbrace{\sum_{y_k \in \mathcal{Y}_k} \exp(-E_B(y_j, y_k))}_{r_{B \to Y_j}(y_j)} \underbrace{\sum_{y_l \in \mathcal{Y}_l} \exp(-E_C(y_j, y_l))}_{r_{C \to Y_j}(y_j)} \\
 &= \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \exp(-E_A(y_i, y_j)) \underbrace{r_{B \to Y_j}(y_j) r_{C \to Y_j}(y_j)}_{q_{Y_j \to A}(y_j)} \\
 &= \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \exp(-E_A(y_i, y_j)) q_{Y_j \to A}(y_j)
 \end{aligned}$$

# Inference on trees (cont.)

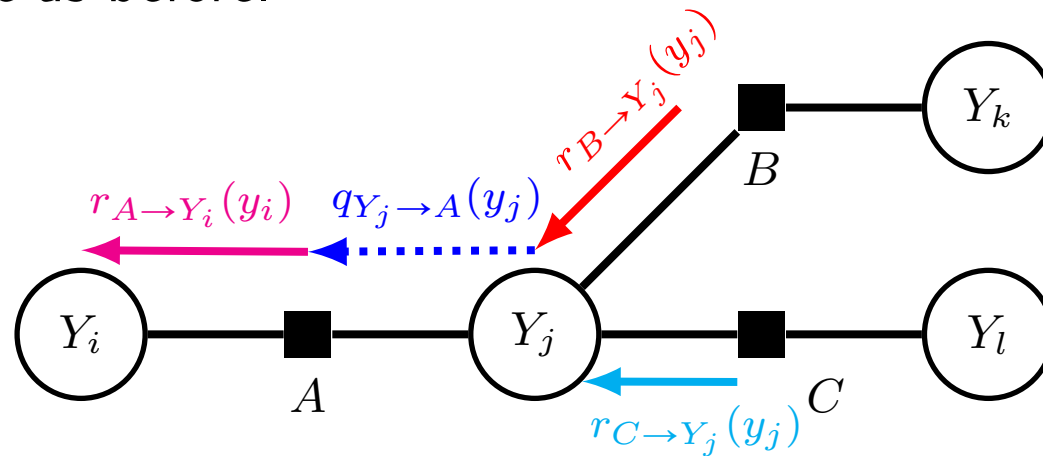


Sum-product algorithm

Max-sum algorithm

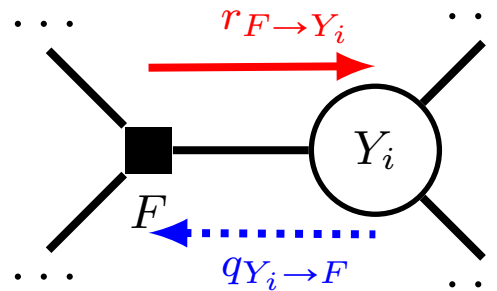
Loopy belief propagation

Now we are assuming a tree-structured factor graph and applying the same elimination procedure as before.



$$\begin{aligned}
 Z &= \sum_{y_i \in \mathcal{Y}_i} \underbrace{\sum_{y_j \in \mathcal{Y}_j} \exp(-E_A(y_i, y_j)) q_{Y_j \rightarrow A}(y_j)}_{r_{A \rightarrow Y_i}(y_i)} \\
 &= \sum_{y_i \in \mathcal{Y}_i} r_{A \rightarrow Y_i}(y_i) .
 \end{aligned}$$

**Message:** pair of vectors at each factor graph edge  $(i, F) \in \mathcal{E}'$ .

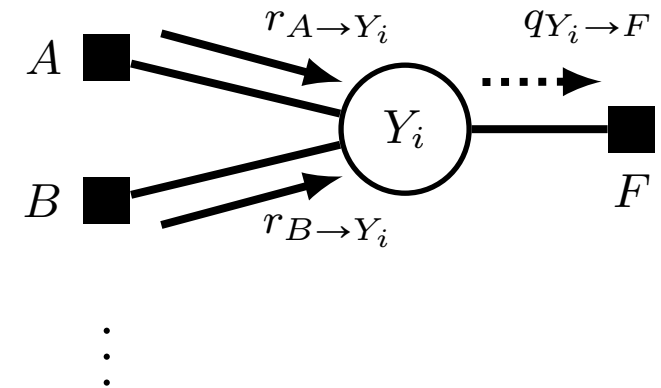


- Variable-to-factor** message  $q_{Y_i \to F} \in \mathbb{R}^{\mathcal{Y}_i}$  is given by

$$q_{Y_i \to F}(y_i) = \prod_{F' \in M(i) \setminus \{F\}} r_{F' \to Y_i}(y_i),$$

where  $M(i) = \{F \in \mathcal{F} : (i, F) \in \mathcal{E}'\}$  denotes the set of factors adjacent to  $Y_i$ .

- Factor-to-variable** message:  $r_{F \to Y_i} \in \mathbb{R}^{\mathcal{Y}_i}$ .



# Factor-to-variable message



Sum-product algorithm

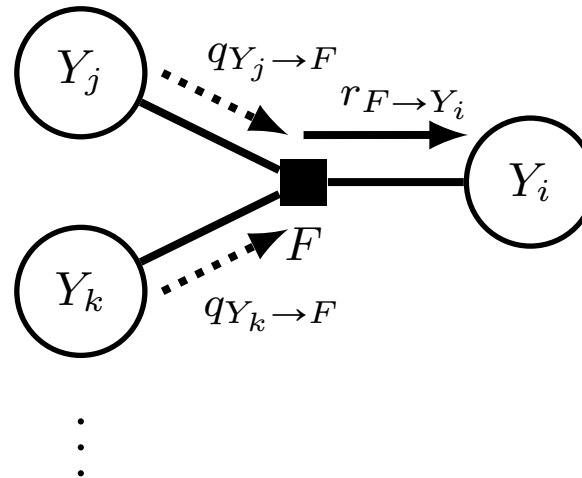
Max-sum algorithm

Loopy belief propagation

2. **Factor-to-variable** message  $r_{F \rightarrow Y_i} \in \mathbb{R}^{\mathcal{Y}_i}$  is given by

$$r_{F \rightarrow Y_i}(y_i) = \sum_{\substack{\mathbf{y}'_F \in \mathcal{Y}_F, \\ y'_i = y_i}} \left( \exp(-E_F(\mathbf{y}'_F)) \prod_{l \in N(F) \setminus \{i\}} q_{Y_l \rightarrow F}(y'_l) \right),$$

where  $N(F) = \{i \in V : (i, F) \in \mathcal{E}'\}$  denotes the set of variables adjacent to  $F$ .



# Message scheduling \*



Sum-product algorithm

Max-sum algorithm

Loopy belief propagation

One can note that the message updates depend on each other.

$$r_{F \rightarrow Y_i}(y_i) = \sum_{\substack{\mathbf{y}'_F \in \mathcal{Y}_F, \\ y'_i = y_i}} \left( \exp(-E_F(\mathbf{y}'_F)) \prod_{l \in N(F) \setminus \{i\}} q_{Y_l \rightarrow F}(y'_l) \right) \quad (1)$$

$$q_{Y_i \rightarrow F}(y_i) = \prod_{F' \in M(i) \setminus \{F\}} r_{F' \rightarrow Y_i}(y_i) \quad (2)$$

The messages that do not depend on previous computation are the following.

- The factor-to-variable messages in which no other variable is adjacent to the factor, that is the product in (1) will be empty.
- The variable-to-factor messages in which no other factor is adjacent to the variable, that is the product in (2) is empty and the message will be one.

# Message scheduling on trees

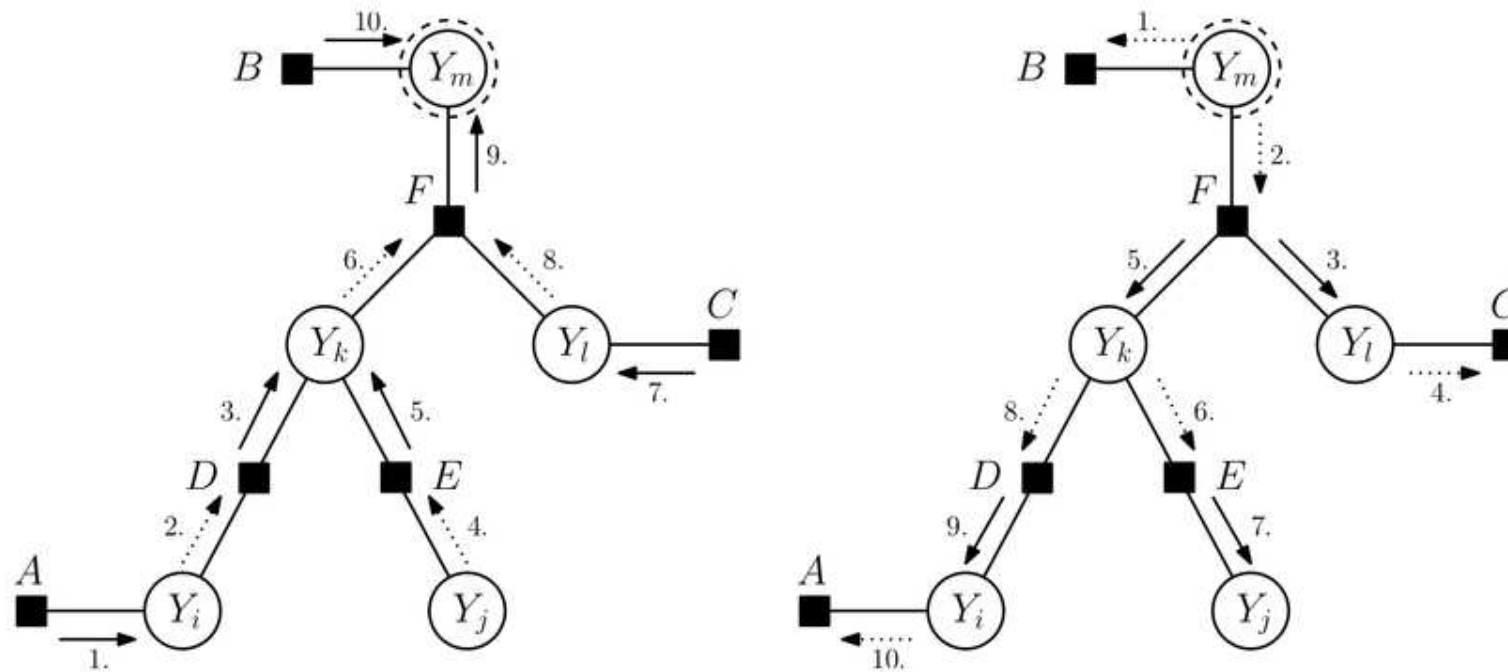


Sum-product algorithm

Max-sum algorithm

Loopy belief propagation

For tree-structured factor graphs there always exist at least one such message that can be computed initially, hence all the dependencies can be resolved.



Source: Nowozin and Lampert. Structured Learning and Prediction. 2011.

1. Select one variable node as root of the tree (e.g.,  $Y_m$ )
2. Compute leaf-to-root messages (e.g., by applying depth-first-search)
3. Compute root-to-leaf messages (reverse order as before)



# Inference result: partition function $Z$



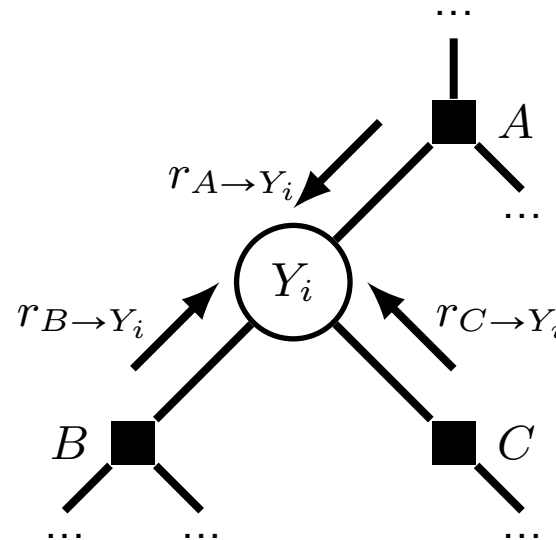
Sum-product algorithm

Max-sum algorithm

Loopy belief propagation

*Partition function* is evaluated at the (root) node  $i$

$$Z = \sum_{y_i \in \mathcal{Y}_i} \prod_{F \in M(i)} r_{F \rightarrow Y_i}(y_i) .$$





# Inference result: the marginals $\mu_F(\mathbf{y}_F)$



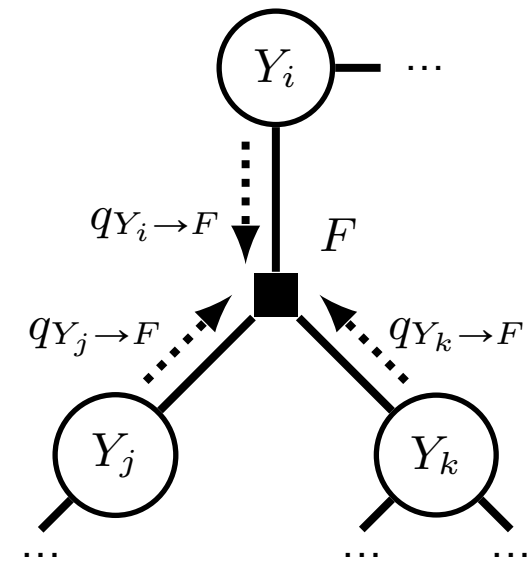
Sum-product algorithm

Max-sum algorithm

Loopy belief propagation

The *marginal distribution* for each factor can be computed as

$$\begin{aligned}\mu_F(\mathbf{y}_F) &\triangleq \sum_{\substack{\mathbf{y}' \in \mathcal{Y}, \\ \mathbf{y}'_F = \mathbf{y}_F}} p(\mathbf{y}) = \sum_{\substack{\mathbf{y}' \in \mathcal{Y}, \\ \mathbf{y}'_F = \mathbf{y}_F}} \frac{1}{Z} \exp\left(-\sum_{H \in \mathcal{F}} E_H(\mathbf{y}'_H)\right) \\ &= \frac{1}{Z} \exp(-E_F(\mathbf{y}_F)) \sum_{\substack{\mathbf{y}' \in \\ H \in \mathcal{F} \setminus \{F\}}} \times \mathcal{Y}_H \exp\left(\sum_{H \in \mathcal{F} \setminus \{F\}} -E_H(\mathbf{y}'_H)\right) \\ &= \frac{1}{Z} \exp(-E_F(\mathbf{y}_F)) \prod_{i \in N(F)} q_{Y_i \rightarrow F}(y_i) .\end{aligned}$$



# Optimality and complexity \*



Sum-product algorithm

Max-sum algorithm

Loopy belief propagation

Assume a *tree-structured* factor graph. If the messages are computed based on *depth-first search order* for the *sum-product algorithm*, then it converges after  $2|\mathcal{V}|$  iterations and provides the **exact** marginals.

If  $|\mathcal{Y}_i| \leq K$  for all  $i \in \mathcal{V}$ , then the complexity of the algorithm  $\mathcal{O}(|\mathcal{V}| \cdot K^L)$ , where  $L = \max_{F \in \mathcal{F}} |N(F)|$ .

$$r_{F \rightarrow Y_i}(y_i) = \sum_{\substack{\mathbf{y}'_F \in \mathcal{Y}_F, \\ y'_i = y_i}} \left( \exp(-E_F(\mathbf{y}'_F)) \prod_{j \in N(F) \setminus \{i\}} q_{Y_j \rightarrow F}(y'_j) \right).$$

Note that the complexity of the naïve way is  $\mathcal{O}(K^{|\mathcal{V}|})$ .

*Reminder:* Assuming  $f, g : \mathbb{R} \rightarrow \mathbb{R}$ , the notation  $f(x) = \mathcal{O}(g(x))$  means that there exists  $C > 0$  and  $x_0 \in \mathbb{R}$  such that  $|f(x)| \leq C|g(x)|$  for all  $x > x_0$ .



Sum-product algorithm

Max-sum algorithm

Loopy belief propagation

# Max-sum algorithm

$$\mathbf{y}^* \in \operatorname{argmax}_{\mathbf{y} \in \mathcal{Y}} p(\mathbf{y}) = \operatorname{argmax}_{\mathbf{y} \in \mathcal{Y}} \frac{1}{Z} \tilde{p}(\mathbf{y}) = \operatorname{argmax}_{\mathbf{y} \in \mathcal{Y}} \tilde{p}(\mathbf{y}) .$$

Similar to the *sum-product algorithm* one can obtain the so-called **max-sum algorithm** to solve the above maximization.

By applying the  $\ln$  function, we have

$$\begin{aligned} \ln \max_{\mathbf{y} \in \mathcal{Y}} \tilde{p}(\mathbf{y}) &= \max_{\mathbf{y} \in \mathcal{Y}} \ln \tilde{p}(\mathbf{y}) \\ &= \max_{\mathbf{y} \in \mathcal{Y}} \ln \prod_{F \in \mathcal{F}} \exp(-E_F(\mathbf{y}_F)) \\ &= \max_{\mathbf{y} \in \mathcal{Y}} \sum_{F \in \mathcal{F}} -E_F(\mathbf{y}_F) . \end{aligned}$$

# MAP inference on trees

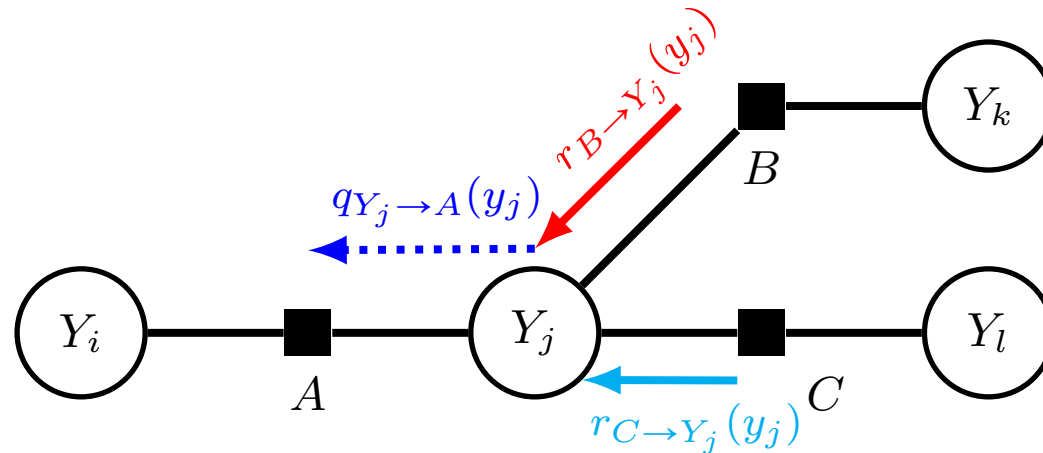


Sum-product algorithm

Max-sum algorithm

Loopy belief propagation

Now we are assuming a tree-structured factor graph and applying an elimination procedure as before.



$$\begin{aligned}
 \max_{\mathbf{y} \in \mathcal{Y}} \sum_{F \in \mathcal{F}} -E_F(\mathbf{y}_F) &= \max_{\mathbf{y} \in \mathcal{Y}} -E_A(y_i, y_j) - E_B(y_j, y_k) - E_C(y_j, y_l) \\
 &= \max_{y_i, y_j} -E_A(y_i, y_j) + \underbrace{\max_{y_k} -E_B(y_j, y_k)}_{r_{B \to Y_j}(y_j)} + \underbrace{\max_{y_l} -E_C(y_j, y_l)}_{r_{C \to Y_j}(y_j)} \\
 &= \max_{y_i, y_j} -E_A(y_i, y_j) + \underbrace{r_{B \to Y_j}(y_j) + r_{C \to Y_j}(y_j)}_{q_{Y_j \to A}(y_j)}
 \end{aligned}$$

# MAP inference on trees (cont.)

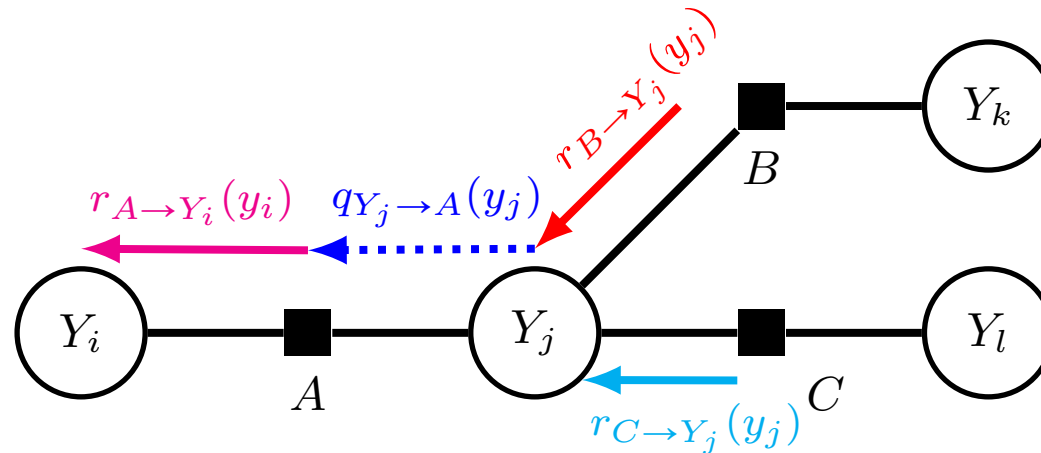


Sum-product algorithm

Max-sum algorithm

Loopy belief propagation

Now we are assuming a tree-structured factor graph and applying an elimination procedure as before.



$$\max_{\mathbf{y} \in \mathcal{Y}} \sum_{F \in \mathcal{F}} -E_F(\mathbf{y}_F) = \max_{y_i} \underbrace{\max_{y_j} -E_A(y_i, y_j) + q_{Y_j \to A}(y_j)}_{r_{A \to Y_i}(y_i)} = \max_{y_i} r_{A \to Y_i}(y_i)$$

The solution is then obtained as:

$$y_i^* \in \operatorname{argmax}_{y_i} r_{A \to Y_i}(y_i), \quad y_j^* \in \operatorname{argmax}_{y_j} -E_A(y_i^*, y_j) + q_{Y_j \to A}(y_j),$$

$$y_k^* \in \operatorname{argmax}_{y_k} -E_B(y_j^*, y_k), \quad y_l^* \in \operatorname{argmax}_{y_l} -E_C(y_j^*, y_l).$$

The messages become as follows

$$q_{Y_i \rightarrow F}(y_i) = \sum_{F' \in M(i) \setminus \{F\}} r_{F' \rightarrow Y_i}(y_i)$$
$$r_{F \rightarrow Y_i}(y_i) = \max_{\substack{y'_F \in \mathcal{Y}_F, \\ y'_i = y_i}} \left( -E_F(y'_F) + \sum_{l \in N(F) \setminus \{i\}} q_{Y_l \rightarrow F}(y'_l) \right).$$

The *max-sum algorithm* provides **exact** MAP inference for tree-structured factor graphs.



# Choosing an optimal state



Sum-product algorithm

Max-sum algorithm

Loopy belief propagation

After calculating the messages, the following **back-tracking** algorithm is applied for choosing an optimal  $y^*$ .

1. Initialize the procedure at the root node ( $Y_i$ ) by choosing any

$$y_i^* \in \operatorname{argmax}_{y_i \in \mathcal{Y}_i} \max_{\mathbf{y}' \in \mathcal{Y}, y'_i = y_i} \tilde{p}(\mathbf{y}'),$$

and set  $\mathcal{I} = \{i\}$ .

2. Based on (*reverse*) *depth-first search order*, for each  $j \in \mathcal{V} \setminus \mathcal{I}$

- (a) choose a configuration  $y_j^*$  at the node  $Y_j$  such that

$$y_j^* \in \operatorname{argmax}_{y_j \in \mathcal{Y}_j} \max_{\substack{\mathbf{y}' \in \mathcal{Y}, \\ y'_j = y_j, \\ y'_i = y_i^* \quad \forall i \in \mathcal{I}}} \tilde{p}(\mathbf{y}'),$$

- (b) update  $\mathcal{I} = \mathcal{I} \cup \{j\}$ .



## ■ Sum-product algorithm

$$q_{Y_i \rightarrow F}(y_i) = \prod_{F' \in M(i) \setminus \{F\}} r_{F' \rightarrow Y_i}(y_i)$$
$$r_{F \rightarrow Y_i}(y_i) = \sum_{\substack{y'_F \in \mathcal{Y}_F, \\ y'_i = y_i}} \left( \exp(-E_F(y'_F)) \prod_{l \in N(F) \setminus \{i\}} q_{Y_l \rightarrow F}(y'_l) \right)$$

## ■ Max-sum algorithm

$$q_{Y_i \rightarrow F}(y_i) = \sum_{F' \in M(i) \setminus \{F\}} r_{F' \rightarrow Y_i}(y_i)$$
$$r_{F \rightarrow Y_i}(y_i) = \max_{\substack{y'_F \in \mathcal{Y}_F, \\ y'_i = y_i}} \left( -E_F(y'_F) + \sum_{l \in N(F) \setminus \{i\}} q_{Y_l \rightarrow F}(y'_l) \right)$$

# Example \*

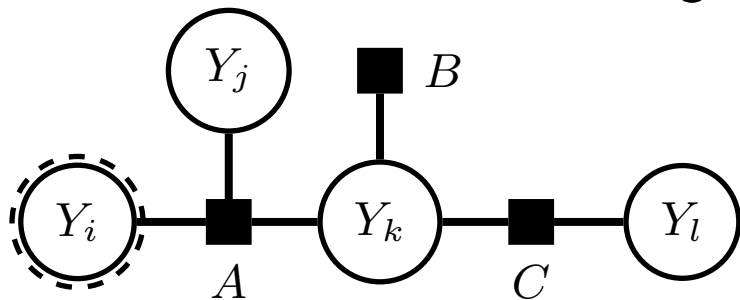


Sum-product algorithm

Max-sum algorithm

Loopy belief propagation

Let us consider the following factor graph with binary variables:



		$E_A(0, y_j, y_k)$		$E_A(1, y_j, y_k)$	
		$y_k$		$y_k$	
$y_j$	0	0	1	0	-1
	1	1	0	0	0

		$E_B(y_k)$
		$y_k$
0	1	
1	0.5	

		$E_C(y_k, y_l)$	
		$y_l$	
$y_k$	0	0	0.5
	1	0.5	0

Let us choose the node  $Y_i$  as root. We calculate the messages for the *max-sum algorithm* from leaf-to-root direction in a *depth-first search order* as follows.

- $q_{Y_l \rightarrow C}(0) = q_{Y_l \rightarrow C}(1) = 0$
- $r_{C \rightarrow Y_k}(0) = \max_{y_l \in \{0,1\}} \{-E_C(0, y_l) + q_{Y_l \rightarrow C}(0)\} = \max_{y_l \in \{0,1\}} -E_C(0, y_l) = 0$   
 $r_{C \rightarrow Y_k}(1) = \max_{y_l \in \{0,1\}} \{-E_C(1, y_l) + q_{Y_l \rightarrow C}(1)\} = \max_{y_l \in \{0,1\}} -E_C(1, y_l) = 0$
- $r_{B \rightarrow Y_k}(0) = -1$   
 $r_{B \rightarrow Y_k}(1) = -0.5$
- $q_{Y_k \rightarrow A}(0) = r_{B \rightarrow Y_k}(0) + r_{C \rightarrow Y_k}(0) = -1 + 0 = -1$   
 $q_{Y_k \rightarrow A}(1) = r_{B \rightarrow Y_k}(1) + r_{C \rightarrow Y_k}(1) = -0.5 + 0 = -0.5$

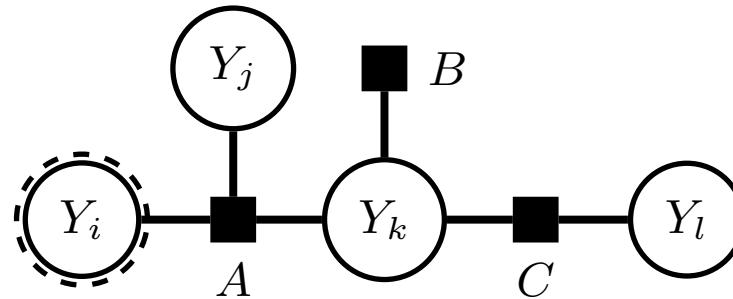
# Example (cont.) \*



Sum-product algorithm

Max-sum algorithm

Loopy belief propagation



5.  $q_{Y_j \rightarrow A}(0) = q_{Y_j \rightarrow A}(1) = 0$
6.  $r_{A \rightarrow Y_i}(0) = \max_{y_j, y_k \in \{0,1\}} \{-E_A(0, y_j, y_k) + q_{Y_j \rightarrow A}(y_j) + q_{Y_k \rightarrow A}(y_k)\} = -0.5$   
 $r_{A \rightarrow Y_i}(1) = \max_{y_j, y_k \in \{0,1\}} \{-E_A(1, y_j, y_k) + q_{Y_j \rightarrow A}(y_j) + q_{Y_k \rightarrow A}(y_k)\} = 0.5$

In order to calculate the maximal state  $y^*$  we apply *back-tracking*

1.  $y_i^* \in \operatorname{argmax}_{y_i \in \{0,1\}} r_{A \rightarrow Y_i}(y_i) = \{1\}$
2.  $y_j^* \in \operatorname{argmax}_{y_j} \max_{y_j, y_k \in \{0,1\}} \{-E_A(1, y_j, y_k) + q_{Y_k \rightarrow A}(y_k)\} = \{0\}$
3.  $y_k^* \in \operatorname{argmax}_{y_k \in \{0,1\}} \{-E_A(1, 0, y_k) + r_{B \rightarrow Y_k}(y_k) + r_{C \rightarrow Y_k}(y_k)\} = \{1\}$
4.  $y_l^* \in \operatorname{argmax}_{y_l \in \{0,1\}} \{-E_C(1, y_l) + r_{C \rightarrow Y_k}(1)\} = \{1\}$

Therefore, the optimal state  $y^* = (y_i^*, y_j^*, y_k^*, y_l^*) = (1, 0, 1, 1)$ .



Sum-product algorithm

Max-sum algorithm

Loopy belief propagation

# Loopy belief propagation

# Message passing in cyclic graphs

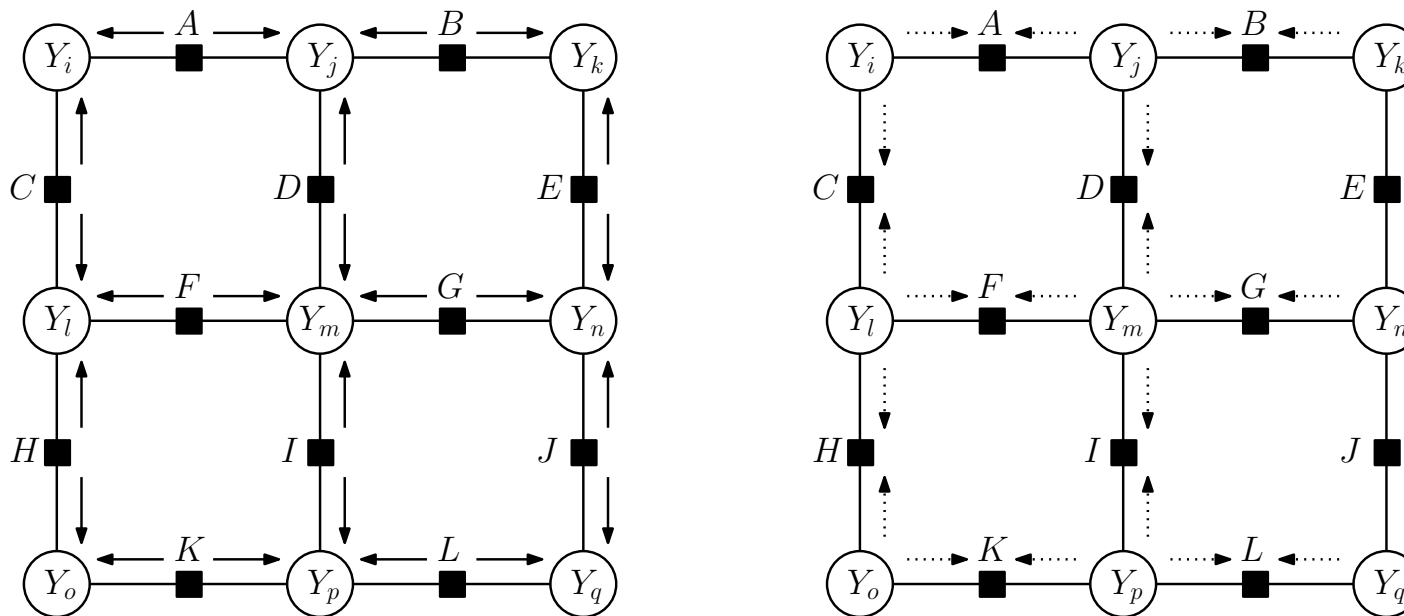


Sum-product algorithm

Max-sum algorithm

Loopy belief propagation

When the graph has cycles, then there is no well-defined *leaf-to-root* order. However, one can apply message passing on cyclic graphs, which results in **loopy belief propagation**.



Source: Nowozin and Lampert. Structured Learning and Prediction. 2011.

1. Initialize all messages as constant 1
2. Pass factor-to-variables and variables-to-factor messages alternately until convergence
3. Upon convergence, treat **beliefs**  $\mu_F$  as approximate marginals

The **factor-to-variable messages**  $r_{F \rightarrow Y_i}$  remain well-defined and are computed as before:

$$r_{F \rightarrow Y_i}(y_i) = \sum_{\substack{\mathbf{y}'_F \in \mathcal{Y}_F, \\ y'_i = y_i}} \left( \exp(-E_F(\mathbf{y}'_F)) \prod_{j \in N(F) \setminus \{i\}} q_{Y_j \rightarrow F}(y'_j) \right).$$

The **variable-to-factor messages** are simply normalized at every iteration as follows:

$$q_{Y_i \rightarrow F}(y_i) = \frac{\prod_{F' \in M(i) \setminus \{F\}} r_{F' \rightarrow Y_i}(y_i)}{\sum_{y'_i \in \mathcal{Y}_i} \prod_{F' \in M(i) \setminus \{F\}} r_{F' \rightarrow Y_i}(y'_i)}.$$

The approximate marginals, i.e. **beliefs**, are computed as before, but now a factor-specific normalization constant  $z_F$  is also used.

The **factor marginals** are given by

$$\mu_F(y_F) = \frac{1}{z_F} \exp(-E_F(y_F)) \prod_{i \in N(F)} q_{Y_i \rightarrow F}(y_i),$$

where the factor specific *normalization constant* is given by

$$z_F = \sum_{y_F \in \mathcal{Y}_F} \exp(-E_F(y_F)) \prod_{i \in N(F)} q_{Y_i \rightarrow F}(y_i).$$



In addition to the factor marginals the algorithm also computes the **variable marginals** in a similar fashion.

$$\mu_i(y_i) = \frac{1}{z_i} \prod_{F' \in M(i)} r_{F' \rightarrow Y_i}(y_i) ,$$

where the normalizing constant is given by

$$z_i = \sum_{y_i \in \mathcal{Y}_i} \prod_{F' \in M(i)} r_{F' \rightarrow Y_i}(y_i) .$$

Since the local *normalization constant*  $z_F$  differs at each factor for loopy belief propagation, the exact value of the normalizing constant  $Z$  **cannot** be directly calculated. Instead, an *approximation to the partition function* can be computed.



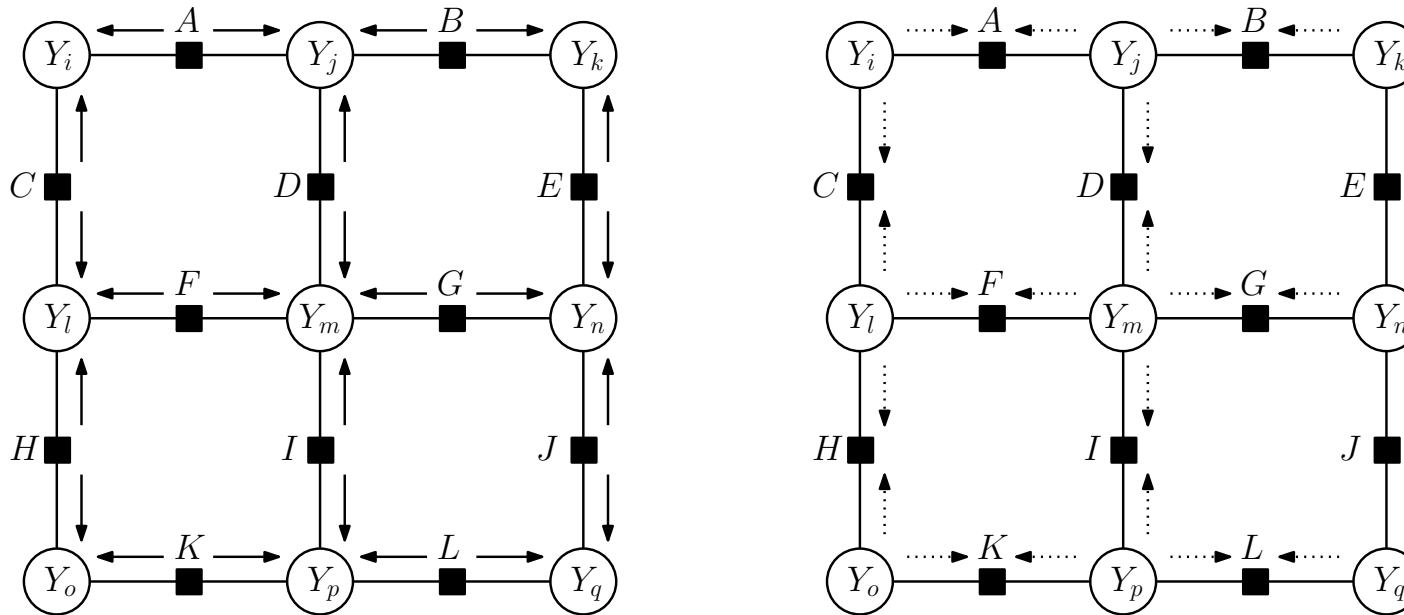
# Remarks on loopy belief propagation



Sum-product algorithm

Max-sum algorithm

Loopy belief propagation



Source: Nowozin and Lampert. Structured Learning and Prediction. 2011.

Loopy belief propagation is very popular, but has some problems:

- It might not converge (e.g., it can oscillate).
- Even if it does, the computed probabilities are only *approximate*.
- If there is a single cycle only in the graph, then it converges.

# Summary \*



Sum-product algorithm

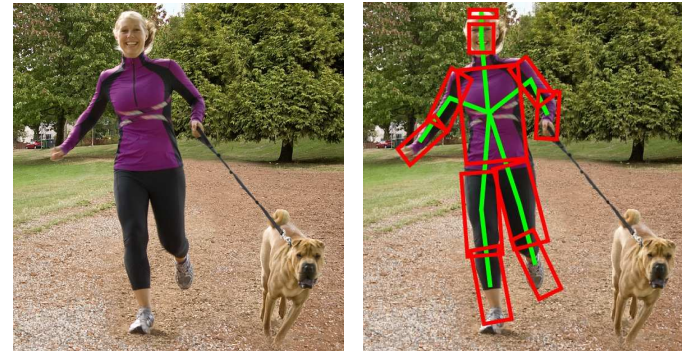
Max-sum algorithm

Loopy belief propagation

- We have discussed **exact inference** methods on *tree-structured* graphical models
  - ◆ Probabilistic inference: *Sum-product algorithm*
  - ◆ MAP inference: *Max-sum algorithm*
- For *general* factor graphs: *Loopy belief propagation*

In the **next lecture** we will learn about

- Human-pose estimation



Source: Nowozin and Lampert. Structured Learning and Prediction. 2011.

- *Mean-field approximation*: probabilistic inference via optimization (a.k.a. variational inference)



# Literature \*



Sum-product algorithm

Max-sum algorithm

Loopy belief propagation

1. Sebastian Nowozin and Christoph H. Lampert. Structured prediction and learning in computer vision. *Foundations and Trends in Computer Graphics and Vision*, 6(3–4), 2010
2. Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009
3. Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Network of Plausible Inference*. Morgan Kaufmann, 1988