NLP 201: Inference In Graphical Models

Jeffrey Flanigan

November 18, 2021

University of California Santa Cruz
jmflanig@ucsc.edu

Many slides and figures from Matt Gormely, David Sontag, Eric Xing, Karteek Alahari, Srijan Kumar, Daniel Khashabi and Noah Smith

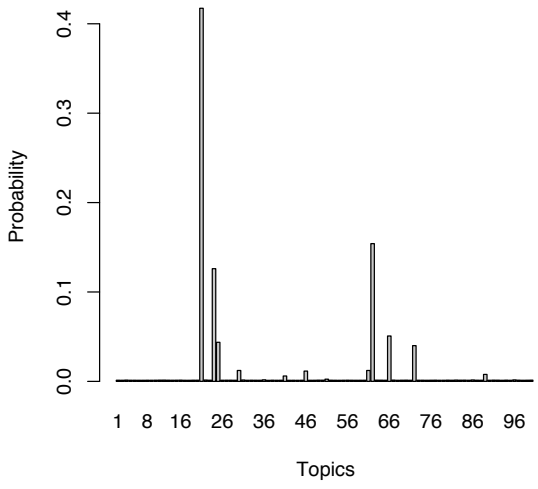
Plan for Today

- Applications of graphical models
- Variable elimination
- Belief propagation (BP)
  - Sum-product BP
  - Max-product BP
  - Loopy BP
- Semi-CRFs

Applications of Graphical Models

- Machine learning
- Computational statistics
- Computer vision and graphics
- Natural language processing
- Informational retrieval
- Robotic control
- Decision making under uncertainty
- Error-control codes
- Computational biology
- Genetics and medical diagnosis/prognosis
- Finance and economics

Example: topic modeling



"Genetics"	"Evolution"	"Disease"	"Computers"
human	evolution	disease	computer
genome	evolutionary	host	models
dna	species	bacteria	information
genetic	organisms	diseases	data
genes	life	resistance	computers
sequence	origin	bacterial	system
gene	biology	new	network
molecular	groups	strains	systems
sequencing	phylogenetic	control	model
map	living	infectious	parallel
information	diversity	malaria	methods
genetics	group	parasite	networks
mapping	new	parasites	software
project	two	united	new
sequences	common	tuberculosis	simulations

For documents in a large collection of text, model $p(\text{Word}|\text{Topic})$, $p(\text{Topic})$

Figure from (Blei, 2011), shows topics and top words learned automatically from reading 17,000 Science articles

Example: image segmentation

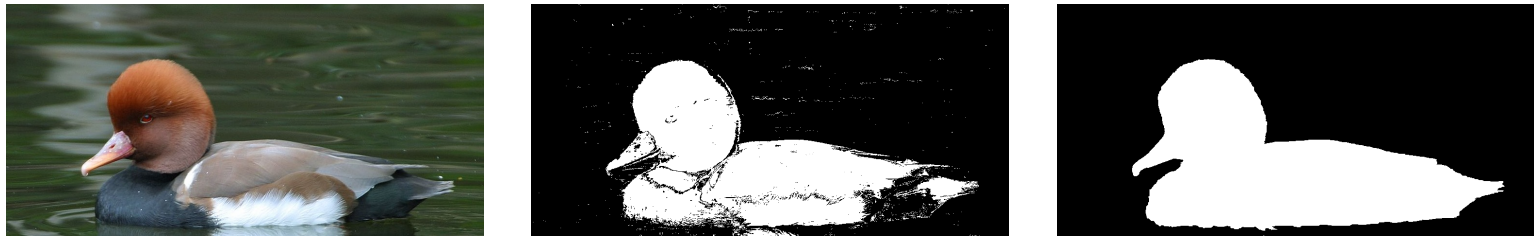
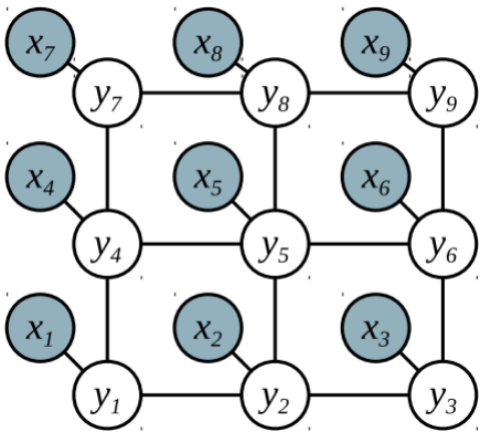


Figure (Nowozin and Lampert, 2012) shows image segmentation problem, original image on left, where goal is to separate foreground from background

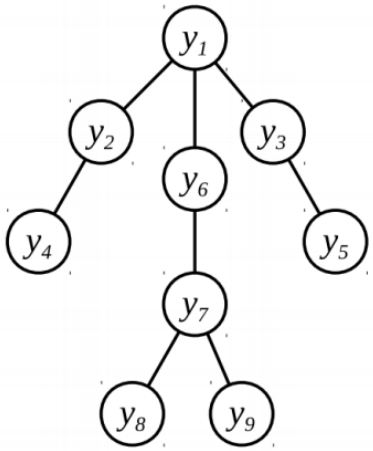
Middle figure shows a segmentation where each pixel is individually classified as belonging to foreground or background

Right figure shows a segmentation where the segmentation is inferred from a probability model over all pixels jointly (encoding probability that neighboring pixels tend to belong to the same group)

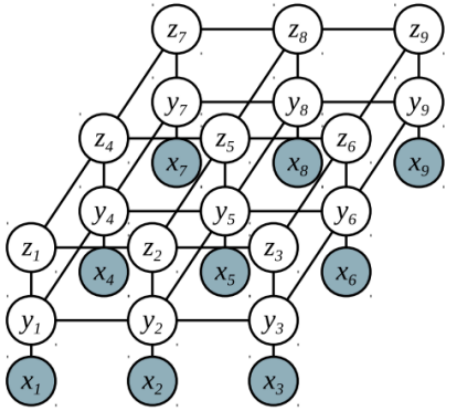
Graphical Models for Computer Vision



Pixel labeling

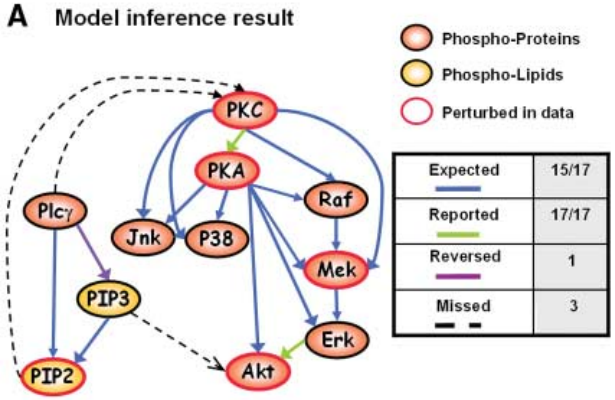


Object detection
Pose estimation



Scene understanding

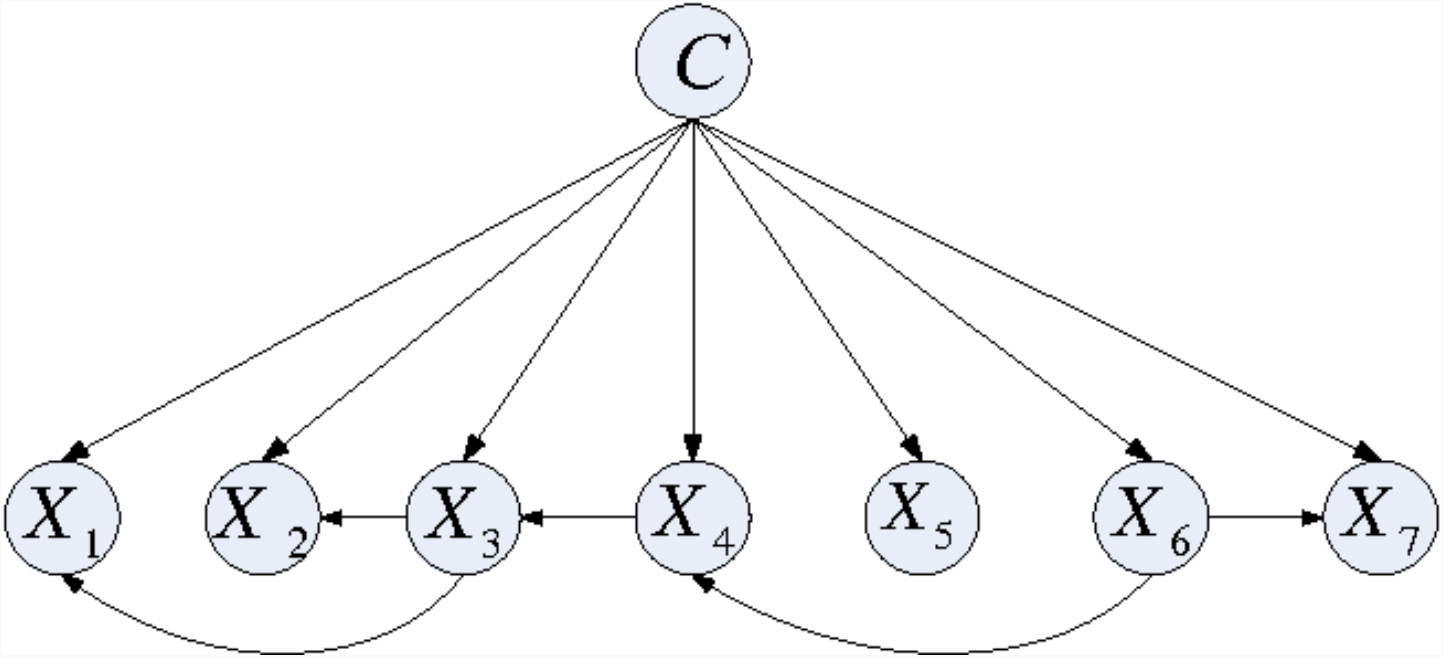
Example: modeling protein networks



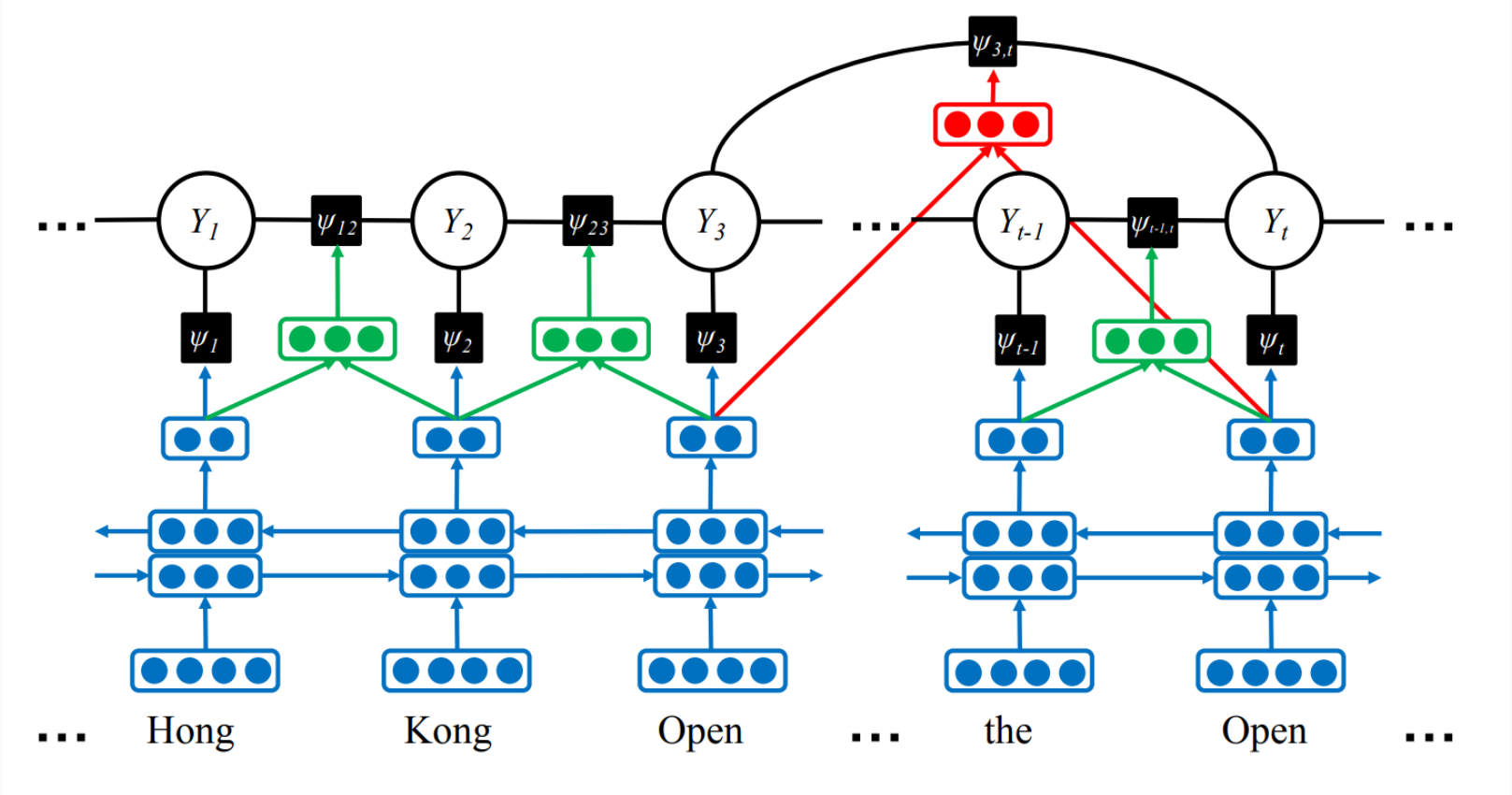
In cellular modeling, can we automatically determine how the presence or absence of some proteins affects other proteins?

Figure from (Sachs et al., 2005) shows automatically inferred protein probability network, which captured most of the known interactions using data-driven methods (far less manual effort than previous methods)

Tree Augmented Naive Bayes Classifier



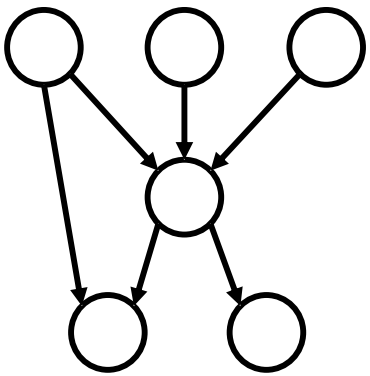
Neural Skip-Chain CRF for Sequence Labeling



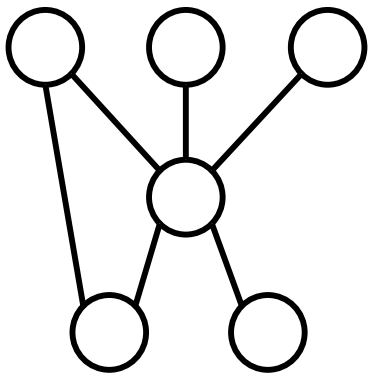
Gao & Gormley (2020)

Three Types of Graphical Models

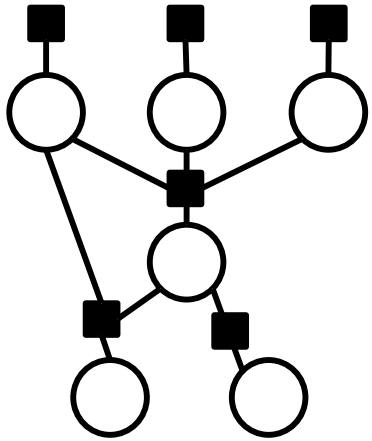
Directed Graphical Model



Undirected Graphical Model



Factor Graph



Three questions for graphical models (Inference)

1. Compute most likely assignment of variables (for prediction)
   MAP Inference (Maximum a posterior inference)

$$\hat{\mathbf{x}} = \operatorname*{argmax}_{\mathbf{x}} p(\mathbf{x}|\theta) \quad \text{or} \quad \hat{\mathbf{y}} = \operatorname*{argmax}_{\mathbf{y}} p(\mathbf{y}|\mathbf{x};\theta)$$

2. Compute normalizer (**partition function**, for learning)

$$Z = \sum_{\mathbf{x}} \prod_{c} \phi_c(\mathbf{x}_c)$$

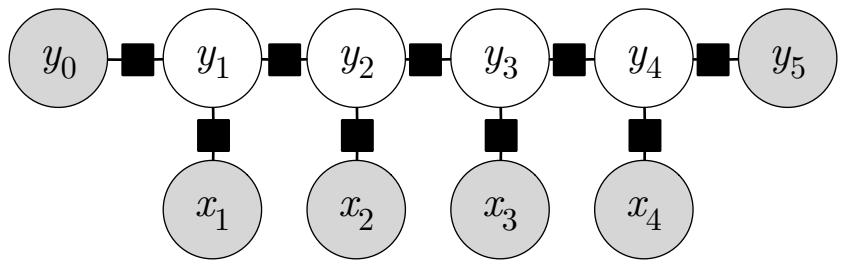
3. Compute marginals (**marginal inference**)

$$p(x_i) = \sum_{\mathbf{x}' : x'_i = x_i} p(\mathbf{x}|\theta)$$

Because factor graphs are the most general graphical model, we will give inference algorithms for factor graphs

We can use them for MRFs, CRFs, and Bayesian Networks

Factor Graphs



- Each box represents a **local factor**, which is a function that depends on the R.V.s it is connected to
- The total score is the product (or sum) of the factors

$$S(\mathbf{x}) = \prod_s \psi_s(\mathbf{x}_s)$$

where s runs over the factors and $\mathbf{x}_s$ denotes the subset of variables in factor s

Factor graphs for CRFs

Simplification:

In a CRF, we only include $\mathbf{y}$ in the graph, and include $\mathbf{x}$ through the factors.

This allows us to use the same algorithms for factor graphs for CRFs

Simple and general exact inference for graphical models

VARIABLE ELIMINATION

Variable Elimination (VE)

VE can compute

- the normalizer (aka partition function): $Z = \sum_{\mathbf{x}'} S(\mathbf{x}')$
- the the marginal of a query variable: $p(x_i) = \sum_{\mathbf{x}' : x'_i = x_i} S(\mathbf{x})$

Let's start with the normalizer

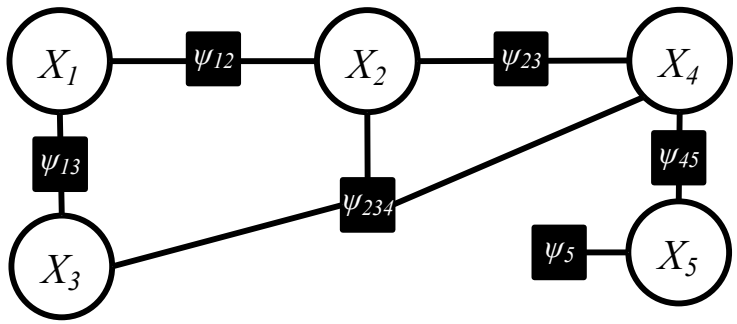
Brute Force (Naïve) Inference

For all i, suppose the **range** of X_i is $\{0, 1, 2\}$.

Let $k=3$ denote the **size of the range**.

The distribution **factorizes** as:

$$s(\boldsymbol{x}) = \psi_{12}(x_1, x_2)\psi_{13}(x_1, x_3)\psi_{24}(x_2, x_4)$$
$$\psi_{234}(x_2, x_3, x_4)\psi_{45}(x_4, x_5)\psi_5(x_5)$$



Naively, we compute the **partition function** as:

$$Z = \sum_{x_1} \sum_{x_2} \sum_{x_3} \sum_{x_4} \sum_{x_5} s(\boldsymbol{x})$$

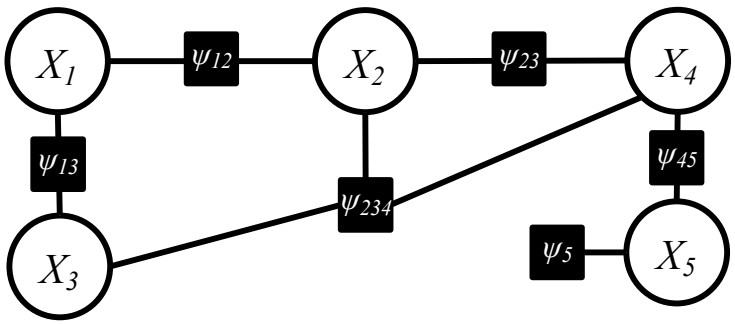
Brute Force (Naïve) Inference

For all i, suppose the **range** of X_i is $\{0, 1, 2\}$.

Let $k=3$ denote the **size of the range**.

The distribution **factorizes** as:

$$s(\boldsymbol{x}) = \psi_{12}(x_1, x_2)\psi_{13}(x_1, x_3)\psi_{24}(x_2, x_4)$$
$$\psi_{234}(x_2, x_3, x_4)\psi_{45}(x_4, x_5)\psi_5(x_5)$$



Naively, we compute the **partition function** as:

$$Z = \sum_{x_1} \sum_{x_2} \sum_{x_3} \sum_{x_4} \sum_{x_5} s(\boldsymbol{x})$$

$s(\boldsymbol{x})$ can be represented as a joint probability table with 3^5 entries:

x_1	x_2	x_3	x_4	x_5	$s(\boldsymbol{x})$
0	0	0	0	0	0.019517693
0	0	0	0	1	0.017090249
0	0	0	0	2	0.014885825
0	0	0	1	0	0.024117638
0	0	0	1	1	0.000925849
0	0	0	1	2	0.028112576
0	0	0	2	0	0.028050205
0	0	0	2	1	0.004812689
0	0	0	2	2	0.007987737
0	0	1	0	0	0.028433687
0	0	1	0	1	0.037073469
0	0	1	0	2	0.013558227
0	0	1	1	0	0.019479016
0	0	1	1	1	0.012312901
0	0	1	1	2	0.023439775
0	0	1	2	0	0.038206131
0	0	1	2	1	0.038996005
0	0	1	2	2	0.041458783
0	0	2	0	0	0.044616806
0	0	2	0	1	0.020846989
0	0	2	0	2	0.03006475
0	0	2	1	0	0.048436964
0	0	2	1	1	0.02854376
0	0	2	1	2	0.029191506
0	0	2	2	0	0.031531118
0	0	2	2	1	0.005132392
0	0	2	2	2	0.032027091
…	…	…	…	…	…

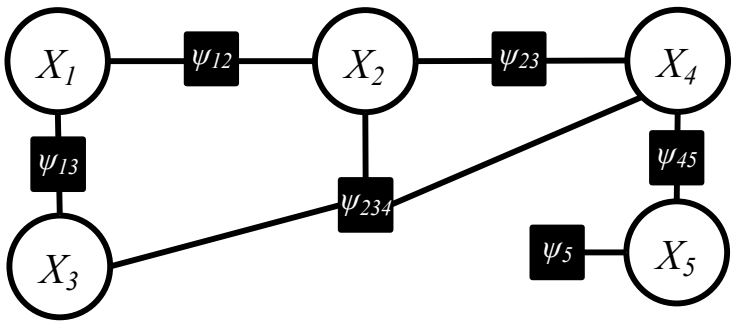
Brute Force (Naïve) Inference

For all i, suppose the **range** of X_i is $\{0, 1, 2\}$.

Let $k=3$ denote the **size of the range**.

The distribution **factorizes** as:

$$s(\boldsymbol{x}) = \psi_{12}(x_1, x_2)\psi_{13}(x_1, x_3)\psi_{24}(x_2, x_4)$$
$$\psi_{234}(x_2, x_3, x_4)\psi_{45}(x_4, x_5)\psi_5(x_5)$$



Naively, we compute the **partition function** as:

$$Z = \sum_{x_1}\sum_{x_2}\sum_{x_3}\sum_{x_4}\sum_{x_5} s(\boldsymbol{x})$$

$s(\boldsymbol{x})$ can be represented as a joint probability table with 3^5 entries:

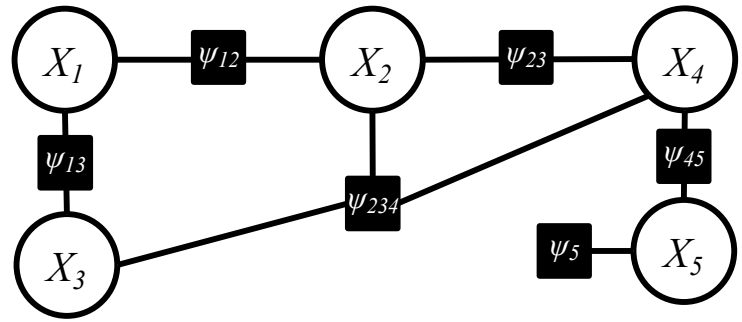
x_1	x_2	x_3	x_4	x_5	$s(\boldsymbol{x})$
0	0	0	0	0	0.019517693
0	0	0	0	1	0.017090249
0	0	0	0	2	0.014885825
0	0	0	1	0	0.024117638
0	0	0	1	1	0.000925849
0	0	0	1	2	0.028112576
0	0	0	2	0	0.028050205
0	0	0	2	1	0.004812689
0	0	0	2	2	0.007987737
0	0	1	0	0	0.028433687
0	0	1	0	1	0.037073469
0	0	1	0	2	0.013558227
0	0	1	1	0	0.019479016
0	0	1	1	1	0.012312901
0	0	1	1	2	0.023439775
0	0	1	2	0	0.038206131
0	0	1	2	1	0.038996005
0	0	1	2	2	0.041458783
0	0	2	0	0	0.044616806
0	0	2	0	1	0.020846989
0	0	2	0	2	0.03006475
0	0	2	1	0	0.048436964
0	0	2	1	1	0.02854376

Naïve computation of Z requires 3^5 additions.

Can we do better?

The Variable Elimination Algorithm

Instead, capitalize on the factorization of $s(\mathbf{x})$.



$$Z = \sum_{x_1}\sum_{x_2}\sum_{x_3}\sum_{x_4}\sum_{x_5} \psi_{12}(x_1,x_2)\psi_{13}(x_1,x_3)\psi_{24}(x_2,x_4)\psi_{234}(x_2,x_3,x_4)\psi_{45}(x_4,x_5)\psi_5(x_5)$$

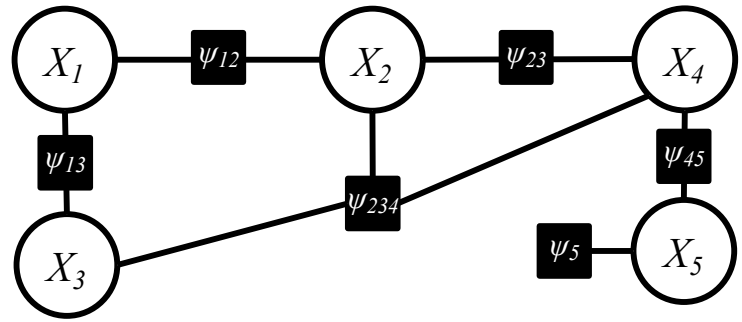
$$= \sum_{x_1}\sum_{x_2}\sum_{x_3}\sum_{x_4} \psi_{12}(x_1,x_2)\psi_{13}(x_1,x_3)\psi_{24}(x_2,x_4)\psi_{234}(x_2,x_3,x_4) \sum_{x_5} \psi_{45}(x_4,x_5)\psi_5(x_5)$$

This "factor" is a much smaller table with 3^2 entries:

x_4	x_5	$s(x_4, x_5)$
0	0	0.019517693
0	1	0.017090249
0	2	0.014885825
1	0	0.024117638
1	1	0.000925849
1	2	0.028112576
2	0	0.028050205
2	1	0.004812689
2	2	0.007987737

The Variable Elimination Algorithm

Instead, capitalize on the factorization of $s(\mathbf{x})$.



$$Z = \sum_{x_1}\sum_{x_2}\sum_{x_3}\sum_{x_4}\sum_{x_5} \psi_{12}(x_1,x_2)\psi_{13}(x_1,x_3)\psi_{24}(x_2,x_4)\psi_{234}(x_2,x_3,x_4)\psi_{45}(x_4,x_5)\psi_5(x_5)$$

$$= \sum_{x_1}\sum_{x_2}\sum_{x_3}\sum_{x_4} \psi_{12}(x_1,x_2)\psi_{13}(x_1,x_3)\psi_{24}(x_2,x_4)\psi_{234}(x_2,x_3,x_4) \sum_{x_5} \psi_{45}(x_4,x_5)\psi_5(x_5)$$

Only 3^2 additions are needed to marginalize out x_5.

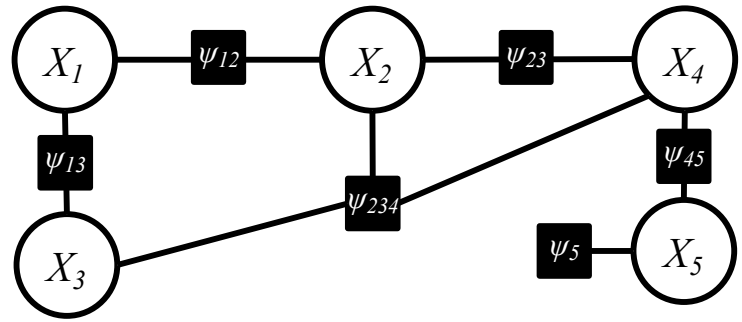
We denote the **marginal's table** by $m_5(x_4)$.

This "factor" is a much smaller table with *3* entries:

x_4	$m_5(x_4)$
0	0.019517693
1	0.017090249
2	0.014885825

The Variable Elimination Algorithm

Instead, capitalize on the factorization of $s(\mathbf{x})$.

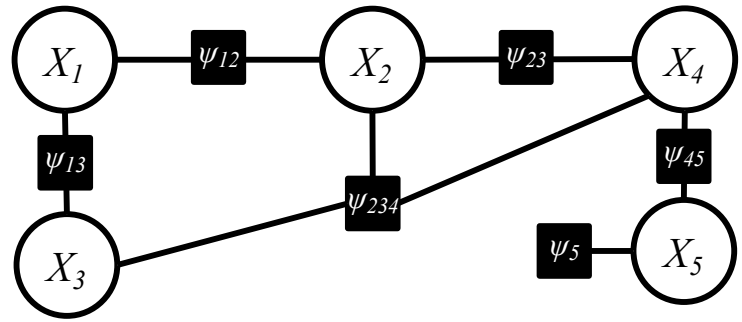


$$
\begin{aligned}
Z &= \sum_{x_1}\sum_{x_2}\sum_{x_3}\sum_{x_4}\sum_{x_5} \psi_{12}(x_1,x_2)\psi_{13}(x_1,x_3)\psi_{24}(x_2,x_4)\psi_{234}(x_2,x_3,x_4)\psi_{45}(x_4,x_5)\psi_5(x_5) \\
&= \sum_{x_1}\sum_{x_2}\sum_{x_3}\sum_{x_4} \psi_{12}(x_1,x_2)\psi_{13}(x_1,x_3)\psi_{24}(x_2,x_4)\psi_{234}(x_2,x_3,x_4) \sum_{x_5} \psi_{45}(x_4,x_5)\psi_5(x_5) \\
&= \sum_{x_1}\sum_{x_2}\sum_{x_3}\sum_{x_4} \psi_{12}(x_1,x_2)\psi_{13}(x_1,x_3)\psi_{24}(x_2,x_4)\psi_{234}(x_2,x_3,x_4) m_5(x_4)
\end{aligned}
$$

$$
m_5(x_4) \triangleq \sum_{x_5} \psi_{45}(x_4,x_5)\psi_5(x_5)
$$

The Variable Elimination Algorithm

Instead, capitalize on the factorization of $s(\mathbf{x})$.



$$Z = \sum_{x_1}\sum_{x_2}\sum_{x_3}\sum_{x_4}\sum_{x_5} \psi_{12}(x_1,x_2)\psi_{13}(x_1,x_3)\psi_{24}(x_2,x_4)\psi_{234}(x_2,x_3,x_4)\psi_{45}(x_4,x_5)\psi_5(x_5)$$

$$= \sum_{x_1}\sum_{x_2}\sum_{x_3}\sum_{x_4} \psi_{12}(x_1,x_2)\psi_{13}(x_1,x_3)\psi_{24}(x_2,x_4)\psi_{234}(x_2,x_3,x_4) \sum_{x_5} \psi_{45}(x_4,x_5)\psi_5(x_5)$$

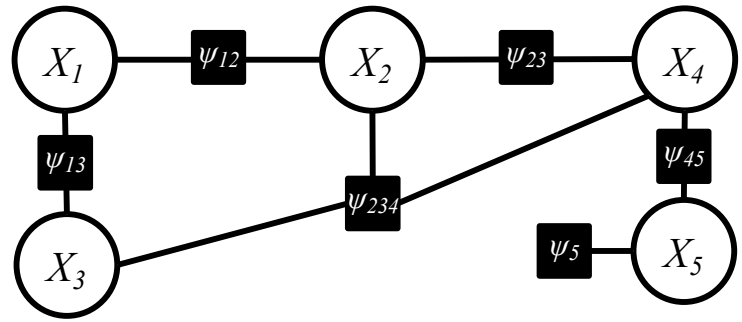
$$= \sum_{x_1}\sum_{x_2}\sum_{x_3}\sum_{x_4} \psi_{12}(x_1,x_2)\psi_{13}(x_1,x_3)\psi_{24}(x_2,x_4)\psi_{234}(x_2,x_3,x_4)m_5(x_4)$$

This "factor" is still a 3^4 table so apply the same trick again.

$$m_5(x_4) \triangleq \sum_{x_5} \psi_{45}(x_4,x_5)\psi_5(x_5)$$

The Variable Elimination Algorithm

Instead, capitalize on the factorization of $s(\mathbf{x})$.



$$Z = \sum_{x_1}\sum_{x_2}\psi_{12}(x_1,x_2)\sum_{x_3}\psi_{13}(x_1,x_3)\sum_{x_4}\psi_{24}(x_2,x_4)\psi_{234}(x_2,x_3,x_4)\sum_{x_5}\psi_{45}(x_4,x_5)\psi_5(x_5)$$

$$= \sum_{x_1}\sum_{x_2}\psi_{12}(x_1,x_2)\sum_{x_3}\psi_{13}(x_1,x_3)\sum_{x_4}\psi_{24}(x_2,x_4)\psi_{234}(x_2,x_3,x_4)m_5(x_4)$$

$$= \sum_{x_1}\sum_{x_2}\psi_{12}(x_1,x_2)\sum_{x_3}\psi_{13}(x_1,x_3)m_4(x_2,x_3)$$

$$= \sum_{x_1}\sum_{x_2}\psi_{12}(x_1,x_2)m_3(x_1,x_2)$$

$$= \sum_{x_1} m_2(x_1)$$

3^2 additions

3^3 additions

3^3 additions

3^2 additions

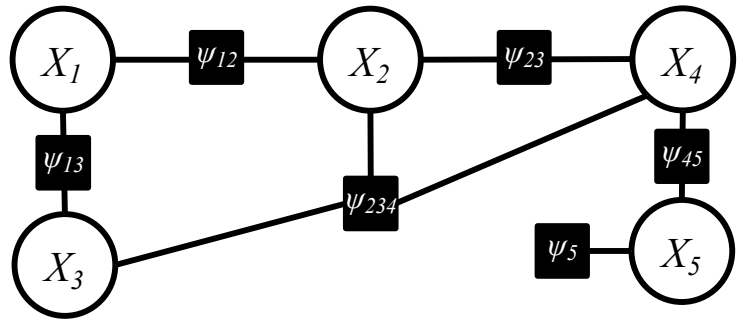
3 additions

Naïve solution requires $3^5=243$ additions.

Variable elimination only requires $3+3^2+3^3+3^3+3^2 = 75$ additions.

The Variable Elimination Algorithm

The same trick can be used to compute **marginal probabilities**. Just choose the variable elimination order such that the query variables are last.



$$p(x_1) = \frac{1}{Z} \sum_{x_2} \psi_{12}(x_1, x_2) \sum_{x_3} \psi_{13}(x_1, x_3) \sum_{x_4} \psi_{24}(x_2, x_4)\psi_{234}(x_2, x_3, x_4) \sum_{x_5} \psi_{45}(x_4, x_5)\psi_5(x_5)$$

$$= \frac{1}{Z} \sum_{x_2} \psi_{12}(x_1, x_2) \sum_{x_3} \psi_{13}(x_1, x_3) \sum_{x_4} \psi_{24}(x_2, x_4)\psi_{234}(x_2, x_3, x_4) m_5(x_4)$$

$$= \frac{1}{Z} \sum_{x_2} \psi_{12}(x_1, x_2) \sum_{x_3} \psi_{13}(x_1, x_3) m_4(x_2, x_3)$$

$$= \frac{1}{Z} \sum_{x_2} \psi_{12}(x_1, x_2) m_3(x_1, x_2)$$

$$= \frac{1}{Z} m_2(x_1)$$

3^2 additions

3^3 additions

3^3 additions

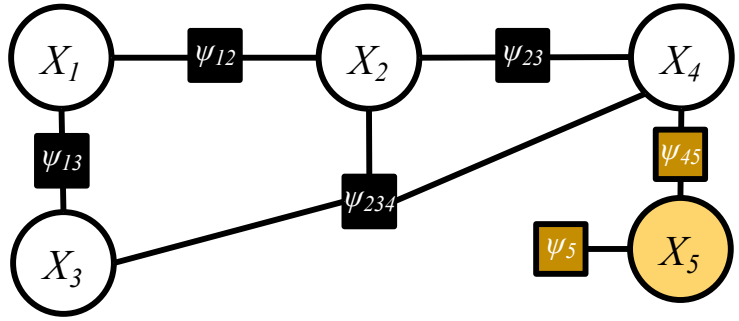
3^2 additions

3 different values on LHS

For directed graphs, Z = 1.

For undirected graphs, if we compute each (unnormalized) value on the LHS, we can sum them to get Z.

The Variable Elimination Algorithm



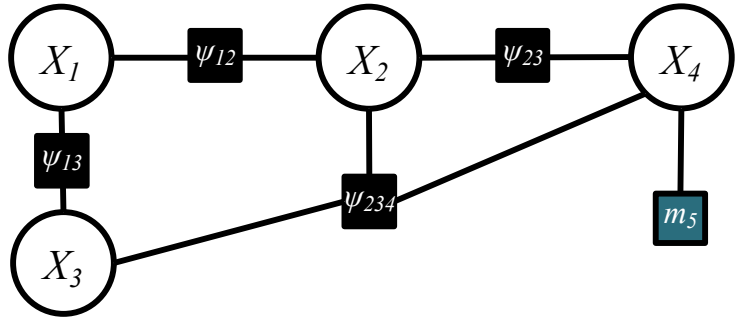
$$Z = \sum_{x_1}\sum_{x_2}\psi_{12}(x_1,x_2)\sum_{x_3}\psi_{13}(x_1,x_3)\sum_{x_4}\psi_{24}(x_2,x_4)\psi_{234}(x_2,x_3,x_4)\sum_{x_5}\psi_{45}(x_4,x_5)\psi_5(x_5)$$

$$= \sum_{x_1}\sum_{x_2}\psi_{12}(x_1,x_2)\sum_{x_3}\psi_{13}(x_1,x_3)\sum_{x_4}\psi_{24}(x_2,x_4)\psi_{234}(x_2,x_3,x_4)m_5(x_4)$$

$$= \sum_{x_1}\sum_{x_2}\psi_{12}(x_1,x_2)\sum_{x_3}\psi_{13}(x_1,x_3)m_4(x_2,x_3)$$

In a factor graph, variable **elimination** corresponds to replacement of a subgraph with a factor.

The Variable Elimination Algorithm



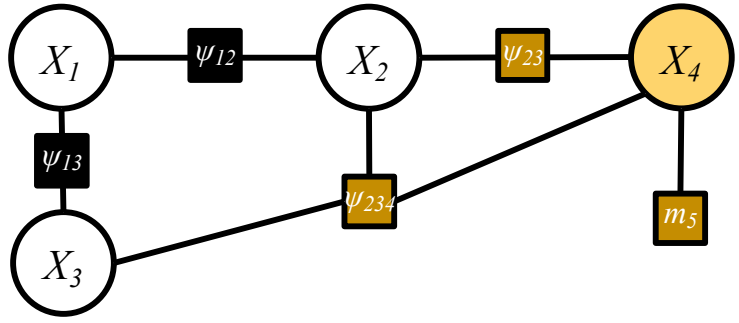
$$Z = \sum_{x_1}\sum_{x_2}\psi_{12}(x_1,x_2)\sum_{x_3}\psi_{13}(x_1,x_3)\sum_{x_4}\psi_{24}(x_2,x_4)\psi_{234}(x_2,x_3,x_4)\sum_{x_5}\psi_{45}(x_4,x_5)\psi_5(x_5)$$

$$= \sum_{x_1}\sum_{x_2}\psi_{12}(x_1,x_2)\sum_{x_3}\psi_{13}(x_1,x_3)\sum_{x_4}\psi_{24}(x_2,x_4)\psi_{234}(x_2,x_3,x_4)m_5(x_4)$$

$$= \sum_{x_1}\sum_{x_2}\psi_{12}(x_1,x_2)\sum_{x_3}\psi_{13}(x_1,x_3)m_4(x_2,x_3)$$

In a factor graph, variable **elimination** corresponds to replacement of a subgraph with a factor.

The Variable Elimination Algorithm



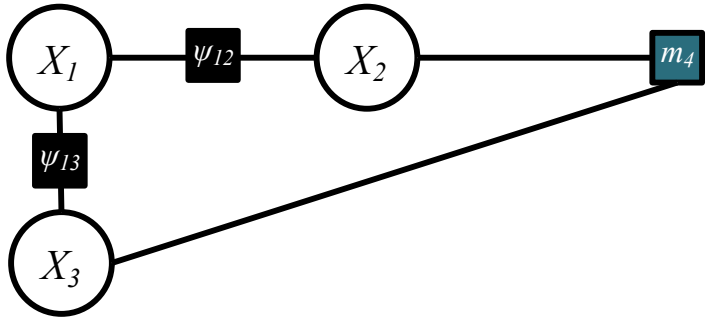
$$Z = \sum_{x_1}\sum_{x_2}\psi_{12}(x_1,x_2)\sum_{x_3}\psi_{13}(x_1,x_3)\sum_{x_4}\psi_{24}(x_2,x_4)\psi_{234}(x_2,x_3,x_4)\sum_{x_5}\psi_{45}(x_4,x_5)\psi_5(x_5)$$

$$= \sum_{x_1}\sum_{x_2}\psi_{12}(x_1,x_2)\sum_{x_3}\psi_{13}(x_1,x_3)\sum_{x_4}\psi_{24}(x_2,x_4)\psi_{234}(x_2,x_3,x_4)m_5(x_4)$$

$$= \sum_{x_1}\sum_{x_2}\psi_{12}(x_1,x_2)\sum_{x_3}\psi_{13}(x_1,x_3)m_4(x_2,x_3)$$

In a factor graph, variable **elimination** corresponds to replacement of a subgraph with a factor.

The Variable Elimination Algorithm



$$Z = \sum_{x_1}\sum_{x_2}\psi_{12}(x_1,x_2)\sum_{x_3}\psi_{13}(x_1,x_3)\sum_{x_4}\psi_{24}(x_2,x_4)\psi_{234}(x_2,x_3,x_4)\sum_{x_5}\psi_{45}(x_4,x_5)\psi_5(x_5)$$

$$= \sum_{x_1}\sum_{x_2}\psi_{12}(x_1,x_2)\sum_{x_3}\psi_{13}(x_1,x_3)\sum_{x_4}\psi_{24}(x_2,x_4)\psi_{234}(x_2,x_3,x_4)m_5(x_4)$$

$$= \sum_{x_1}\sum_{x_2}\psi_{12}(x_1,x_2)\sum_{x_3}\psi_{13}(x_1,x_3)m_4(x_2,x_3)$$

In a factor graph, variable **elimination** corresponds to replacement of a subgraph with a factor.

Variable Elimination for Marginal Inference

Algorithm 1: Variable Elimination for Marginal Inference

Input: the factor graph and the query variable

Output: the marginal distribution for the query variable

a. Run a breadth-first-search starting at the query variable to obtain an ordering of the variable nodes
b. Reverse that ordering
c. Eliminate each variable in the reversed ordering using Algorithm 2

Algorithm 2: Eliminate One Variable

Input: the variable to be eliminated

Output: new factor graph with the variable marginalized out

a. Find the input variable and its neighboring factors -- call this set the eliminated set
b. Replace the eliminated set with a new factor
    a. The neighbors of the new factor should be all the neighbors of all the factors in the eliminated set
    b. The new factor should assign a score to each possible assignment of its neighboring variables
    c. Said score should be identical to the product of the factors it is replacing, summing over the eliminated variable

Variable Elimination for Partition Function

Algorithm 3: Variable Elimination for the Partition Function

Input: the factor graph

Output: the partition function

a. Run a breadth-first-search starting at an arbitrary variable to obtain an ordering of the variable nodes

b. Eliminate each variable in the ordering using Algorithm 2

Algorithm 2: Eliminate One Variable

Input: the variable to be eliminated

Output: new factor graph with the variable marginalized out

a. Find the input variable and its neighboring factors -- call this set the eliminated set

b. Replace the eliminated set with a new factor

   a. The neighbors of the new factor should be all the neighbors of all the factors in the eliminated set

   b. The new factor should assign a score to each possible assignment of its neighboring variables

   c. Said score should be identical to the product of the factors it is replacing, summing over the eliminated variable

Properties of Variable Elimination (VE)

- It is an exact algorithm
- The order of the variable elimination matters
- For a tree, eliminating from the leaves to the root gives a polynomial-time algorithm
- For general graphs, runtime can become **exponential** (large factors are created as you eliminate variables)

Variable Elimination (VE) vs Belief Propagation (BP)

- VE must be run sequentially, BP can be run in parallel
- BP works correctly only for tree-structured models
- BP can be run without changes on graphs with loops (this called Loopy BP), but this is an approximate algorithm
- BP computes all marginals (sum-product version) or all variable assignments (max-product version) at once

Belief Propagation

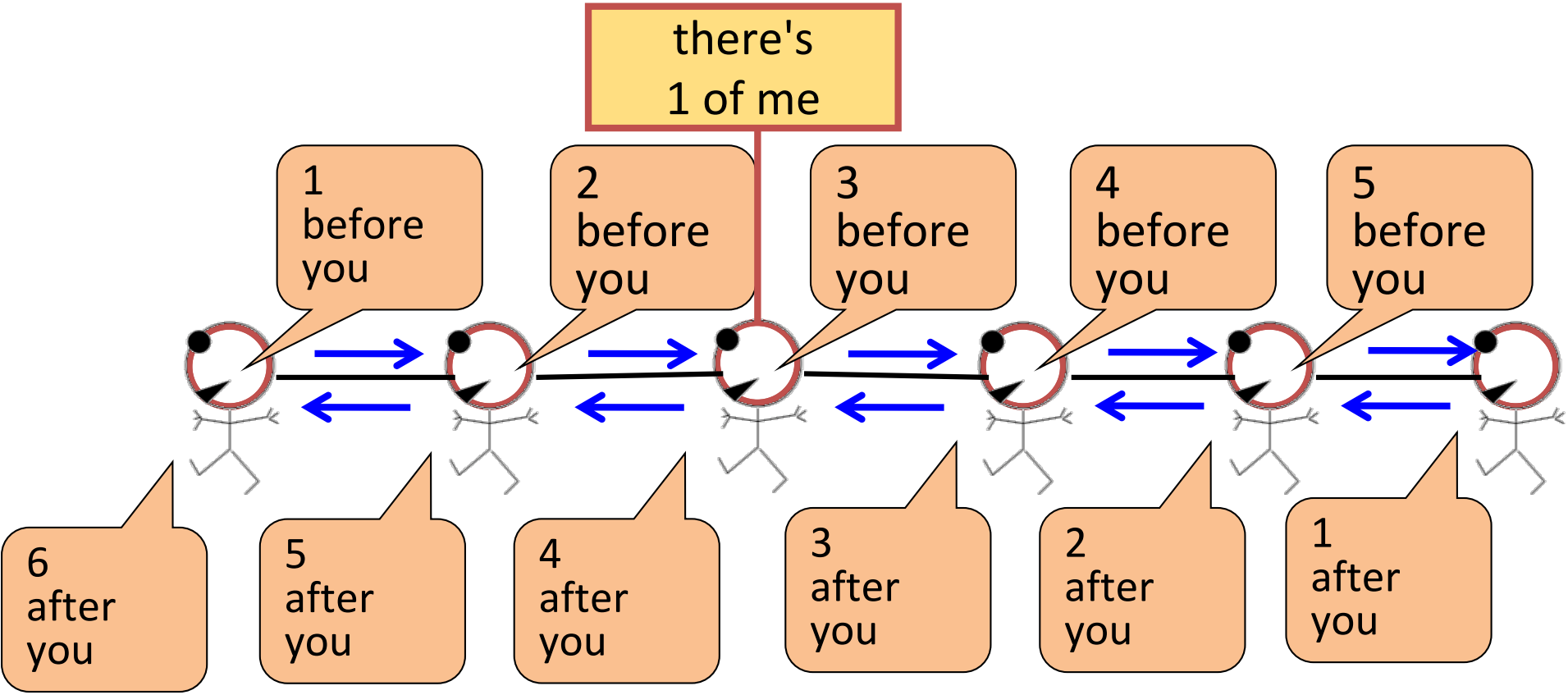
- **Intuition: Use neighbors belief about a node to predict node label**
  - Used to estimate marginals (beliefs) or the most likely states of all variables (nodes)
- Iterative process in which neighbor variables "talk" to each other, passing messages

"I (variable *x1*) believe you (variable *x2*) belong in these states with various likelihoods…"

- When consensus is reached, calculate final belief

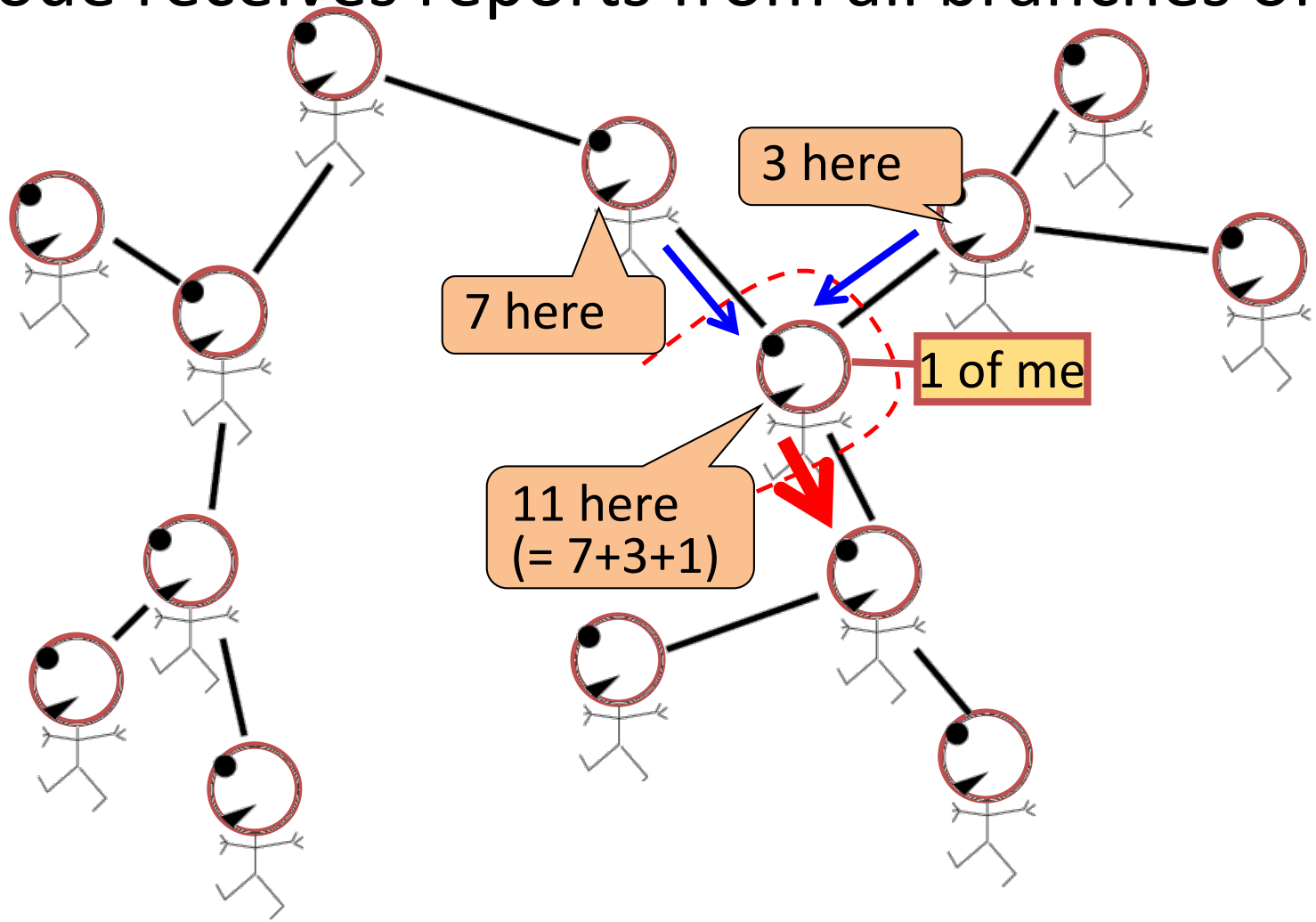
Message Passing Basics

Task: Count the number of nodes in a graph
Condition: Each node can pass message to its neighbors
Solution: Each node listens to the message from its neighbor, updates it, and passes it forward



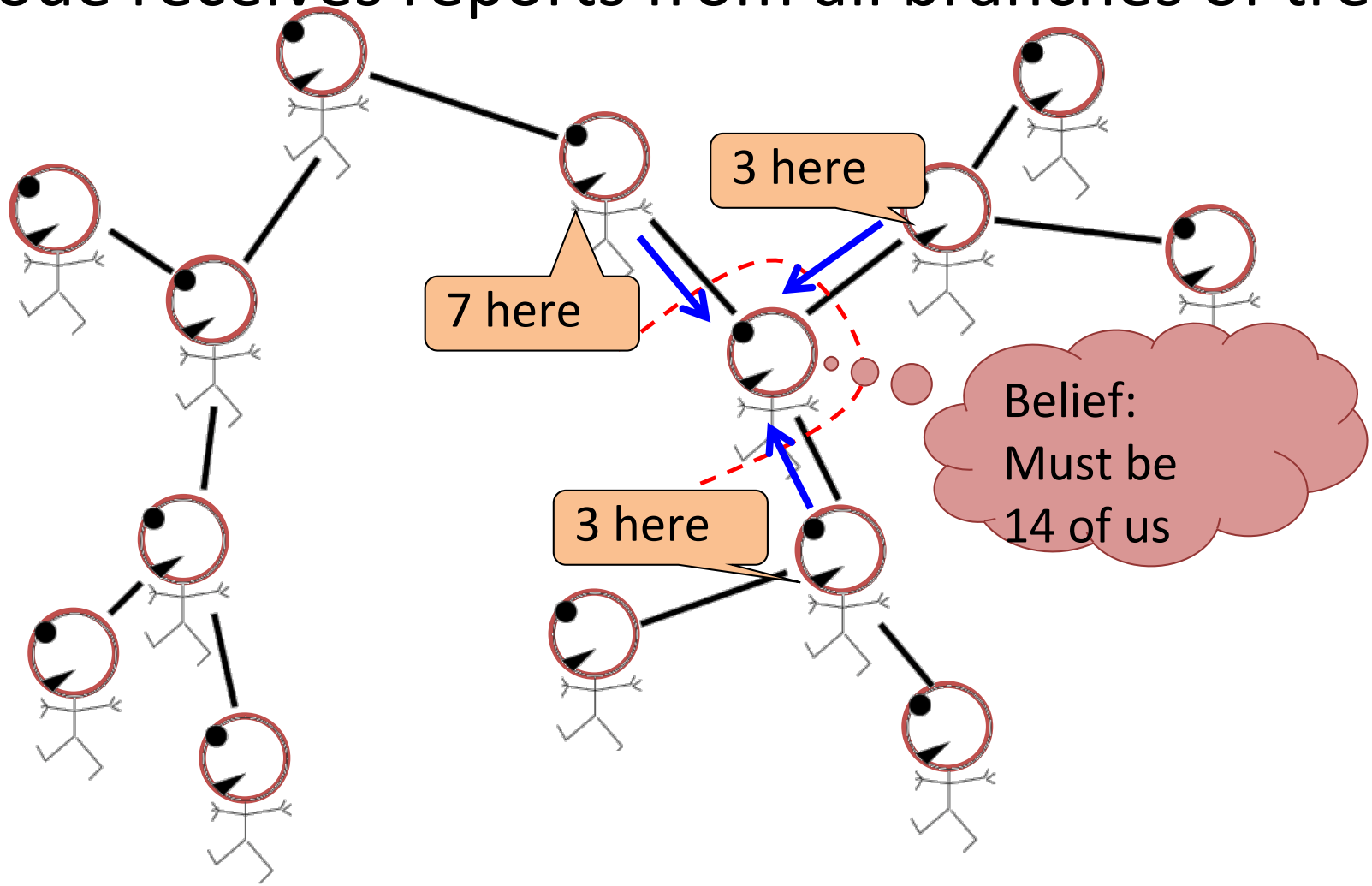
Message Passing in a Tree

Each node receives reports from all branches of tree



Message Passing in a Tree

Each node receives reports from all branches of tree



Running BP Sequentially

To run BP sequentially:

- Pick a root node (any node will work)
- Start at the leaves, and pass the message 1
- Go towards the root node, passing the message of 1 plus the sum of the node's children
- Once you reach the root node, propagate the messages back to the children (pass the message of the node's belief minus the message that the child sent it)

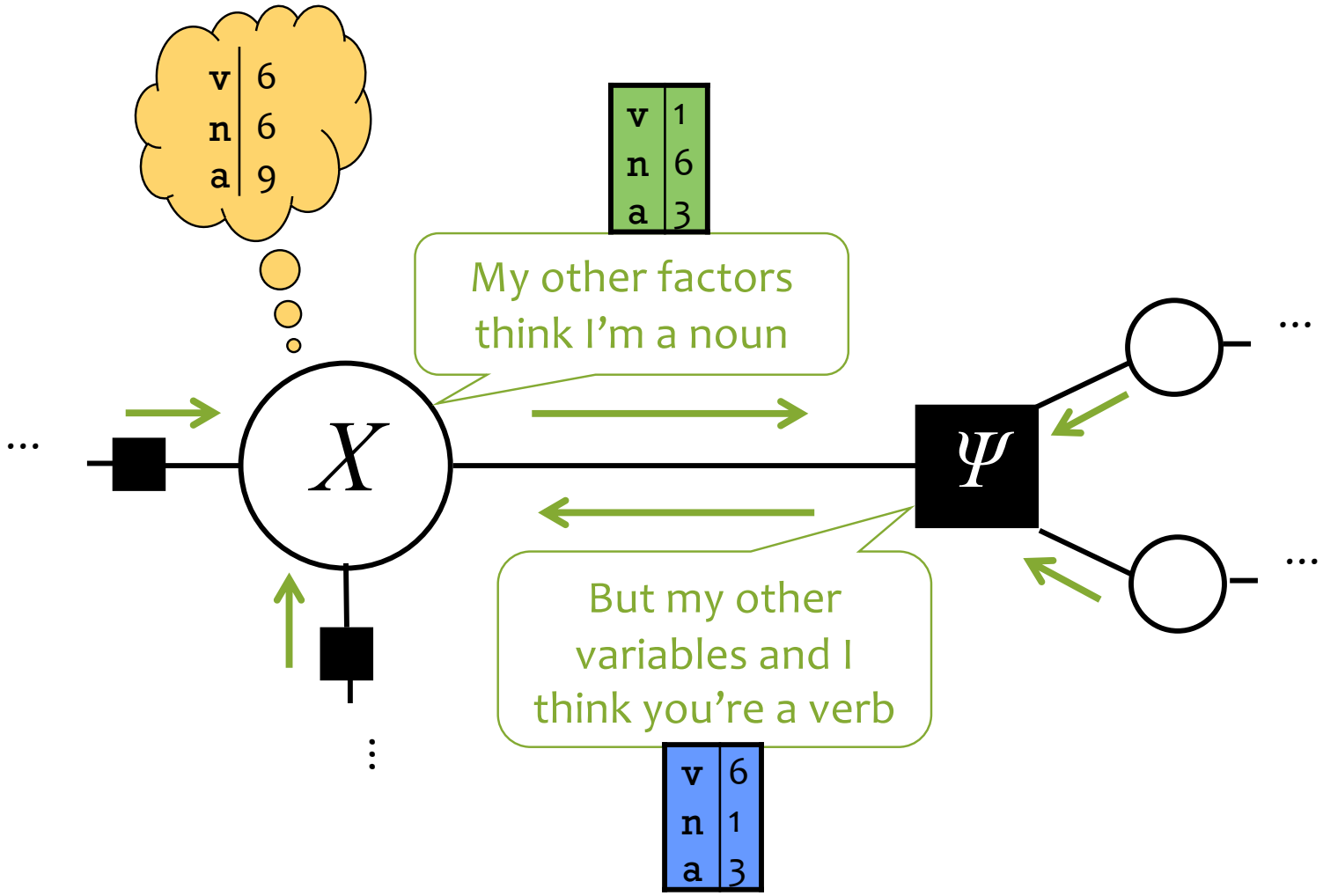
Running BP in Parallel

- You can run BP in parallel (no need to pick a root node)
- Start: Initialize all messages to 1
- At each iteration: all new messages are computed and passed to the the neighbors in parallel
- A node's belief is the sum of the messages from the neighbors plus one
- The message to send from node A to node B is the belief of A minus the message B sent A
- After enough iterations, all the messages will converge, and all the beliefs will be correct (How many iterations? The length of longest path between any two nodes in the graph)

Exact marginal inference for factor trees

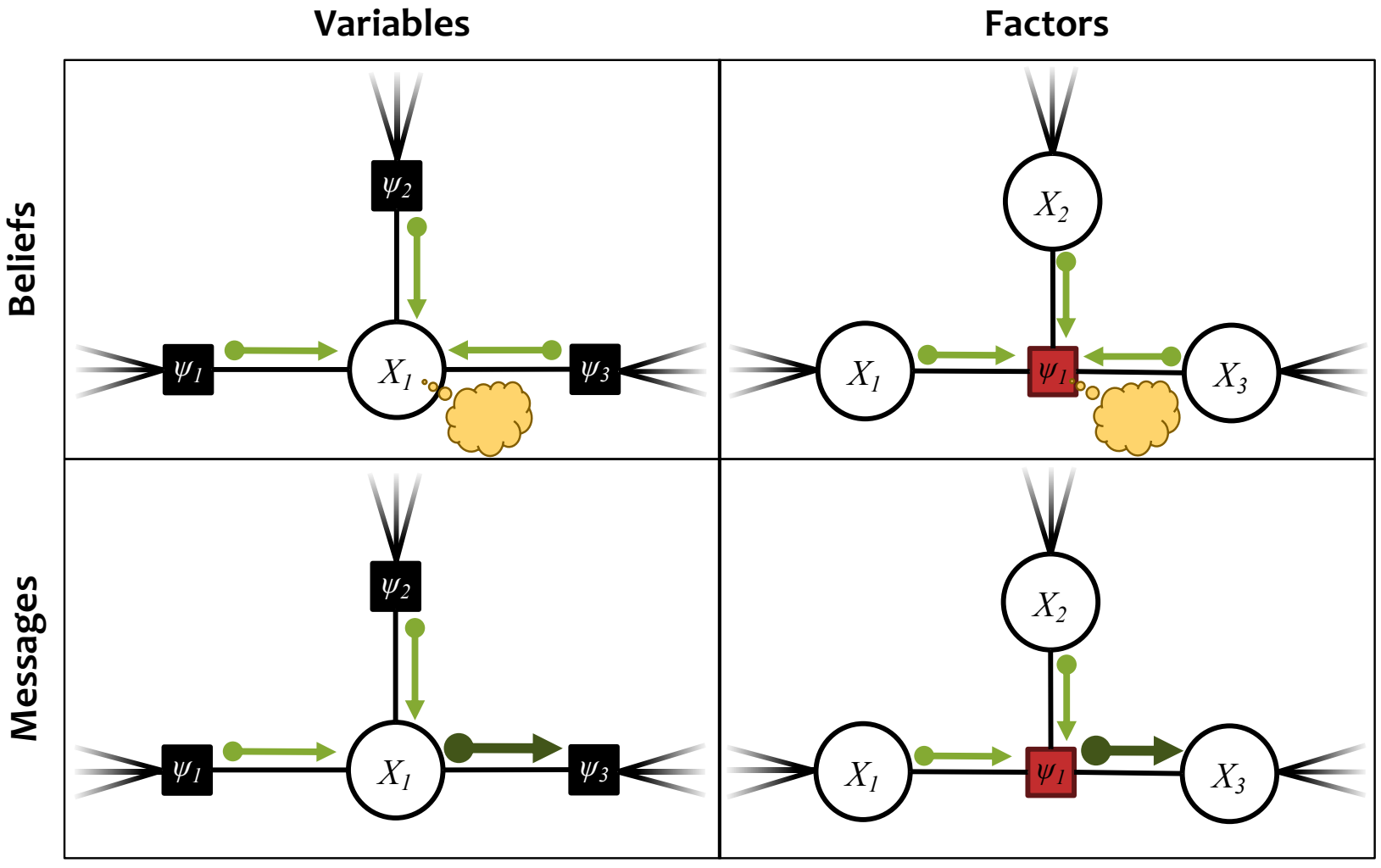
SUM-PRODUCT BELIEF PROPAGATION

Message Passing in Belief Propagation



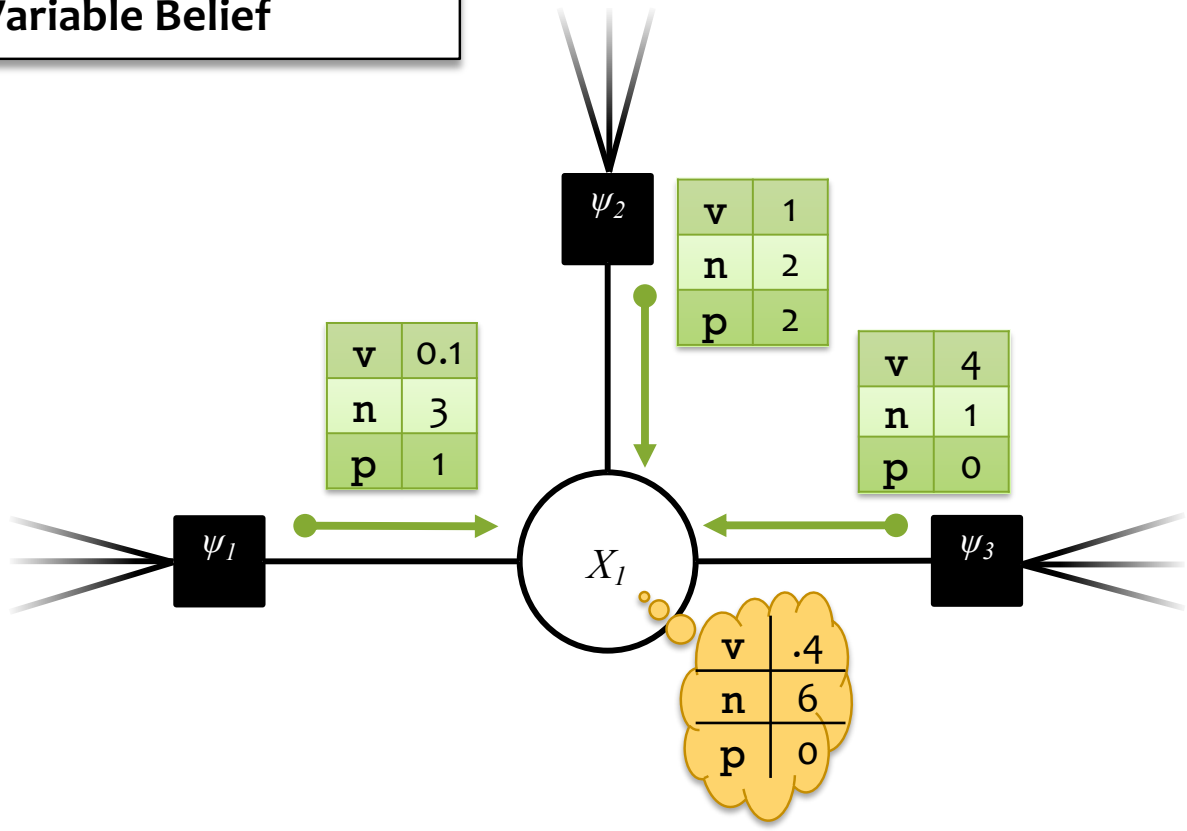
Both of these messages judge the possible values of variable X.
Their product = belief at X = product of all 3 messages to X.

Sum-Product Belief Propagation



Sum-Product Belief Propagation

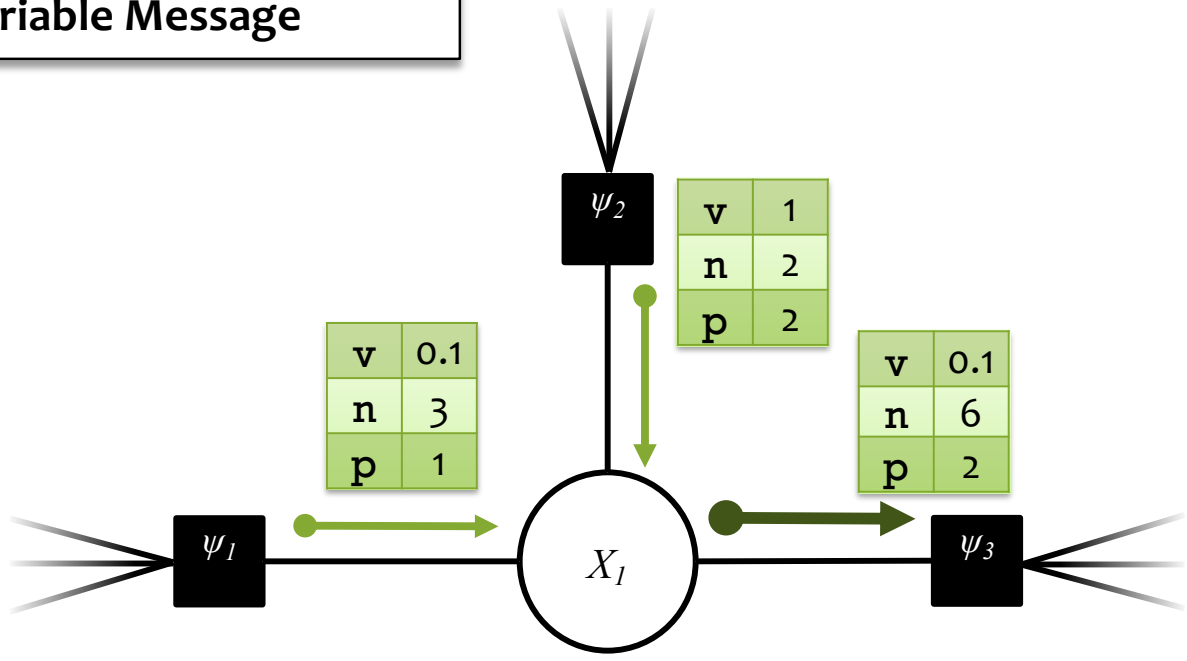
Variable Belief



$$b_i(x_i) = \prod_{\alpha \in \mathcal{N}(i)} \mu_{\alpha \to i}(x_i)$$

Sum-Product Belief Propagation

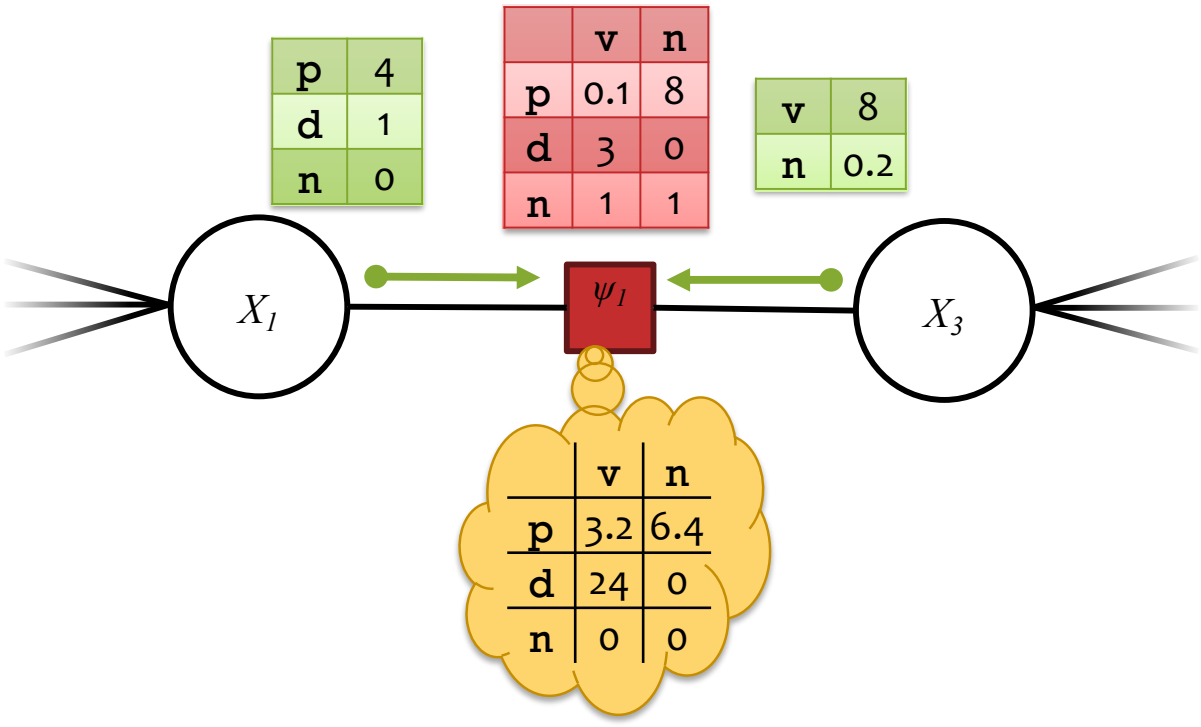
Variable Message



$$\mu_{i \to \alpha}(x_i) = \prod_{\alpha \in \mathcal{N}(i) \setminus \alpha} \mu_{\alpha \to i}(x_i)$$

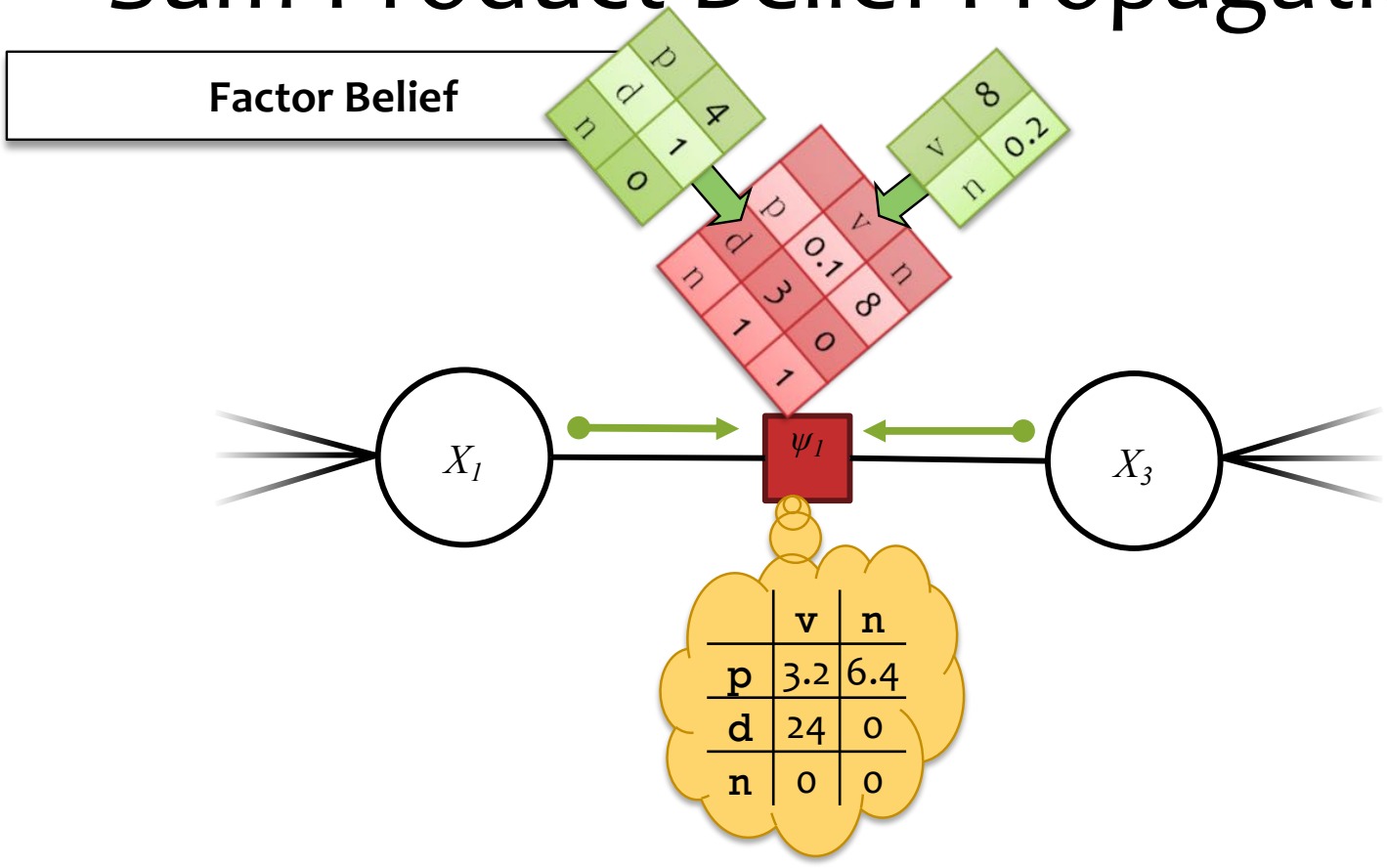
Sum-Product Belief Propagation

Factor Belief



$$b_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}}) = \psi_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}}) \prod_{i \in \mathcal{N}(\alpha)} \mu_{i \to \alpha}(\boldsymbol{x}_{\boldsymbol{\alpha}}[i])$$

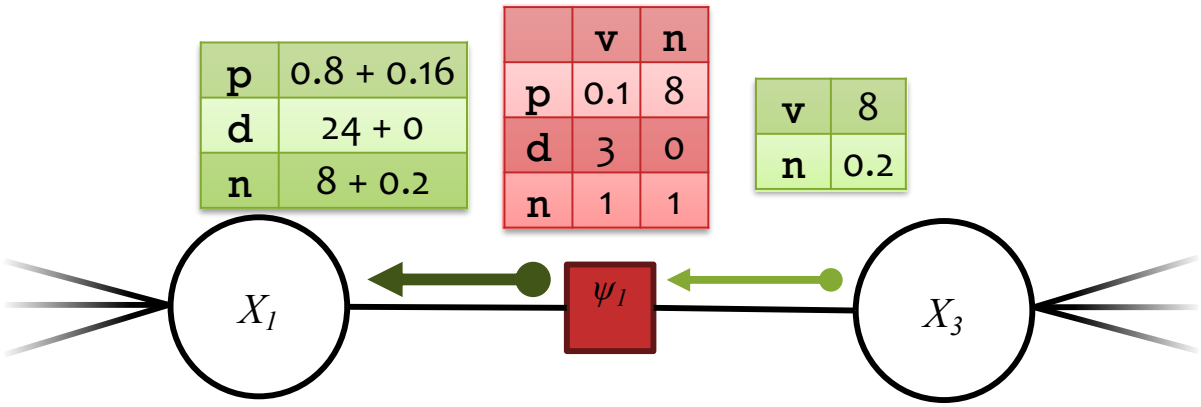
Sum-Product Belief Propagation



$$b_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}}) = \psi_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}}) \prod_{i \in \mathcal{N}(\alpha)} \mu_{i \to \alpha}(\boldsymbol{x}_{\boldsymbol{\alpha}}[i])$$

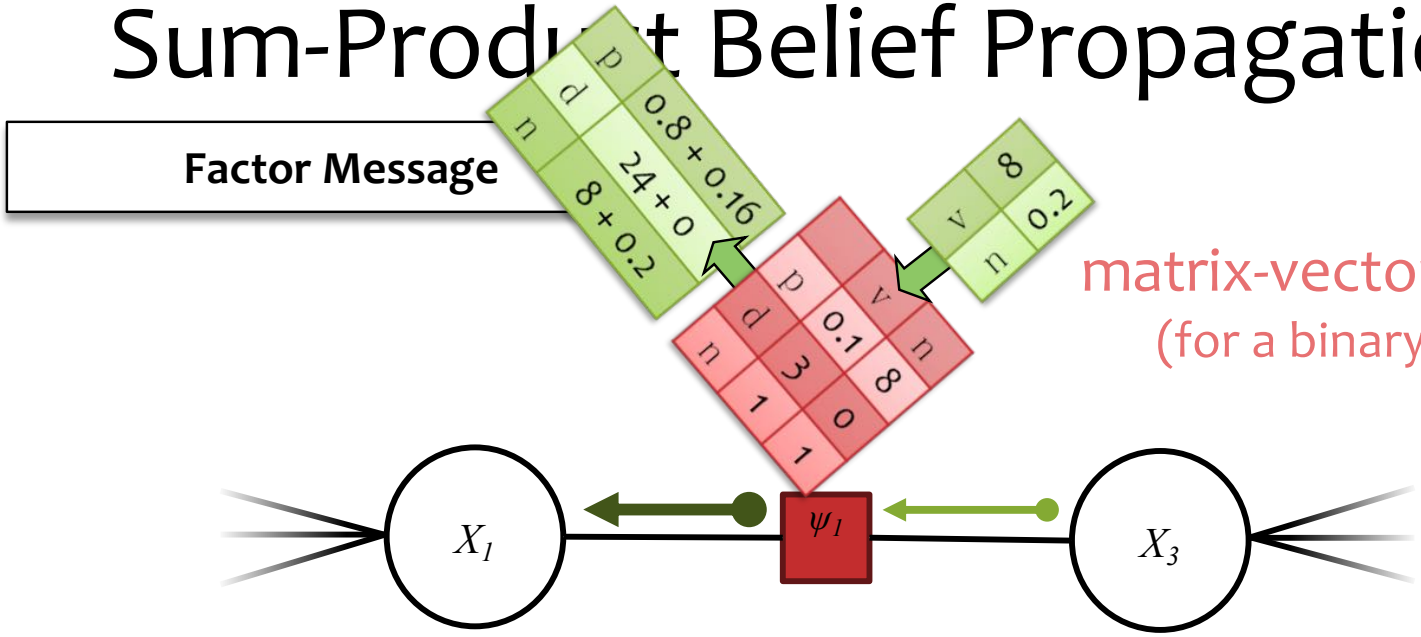
Sum-Product Belief Propagation

Factor Message



$$\mu_{\alpha \to i}(x_i) = \sum_{\boldsymbol{x}_\alpha : \boldsymbol{x}_\alpha[i] = x_i} \psi_\alpha(\boldsymbol{x}_\alpha) \prod_{j \in \mathcal{N}(\alpha) \setminus i} \mu_{j \to \alpha}(\boldsymbol{x}_\alpha[i])$$

Sum-Product Belief Propagation



matrix-vector product
(for a binary factor)

$$\mu_{\alpha \to i}(x_i) = \sum_{\boldsymbol{x}_{\boldsymbol{\alpha}} : \boldsymbol{x}_{\boldsymbol{\alpha}}[i] = x_i} \psi_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}}) \prod_{j \in \mathcal{N}(\alpha) \setminus i} \mu_{j \to \alpha}(\boldsymbol{x}_{\boldsymbol{\alpha}}[i])$$

Sum-Product Belief Propagation

Input: a factor graph with no cycles

Output: exact marginals for each variable and factor

Algorithm:

1. Initialize the messages to the uniform distribution.

$$\mu_{i\to\alpha}(x_i) = 1 \qquad \mu_{\alpha\to i}(x_i) = 1$$

1. Choose a root node.
2. Send messages from the **leaves** to the **root**.
   Send messages from the **root** to the **leaves**.

$$\mu_{i\to\alpha}(x_i) = \prod_{\alpha\in\mathcal{N}(i)\setminus\alpha} \mu_{\alpha\to i}(x_i) \qquad \mu_{\alpha\to i}(x_i) = \sum_{\boldsymbol{x}_{\boldsymbol{\alpha}}:\boldsymbol{x}_{\boldsymbol{\alpha}}[i]=x_i} \psi_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}}) \prod_{j\in\mathcal{N}(\alpha)\setminus i} \mu_{j\to\alpha}(\boldsymbol{x}_{\boldsymbol{\alpha}}[i])$$

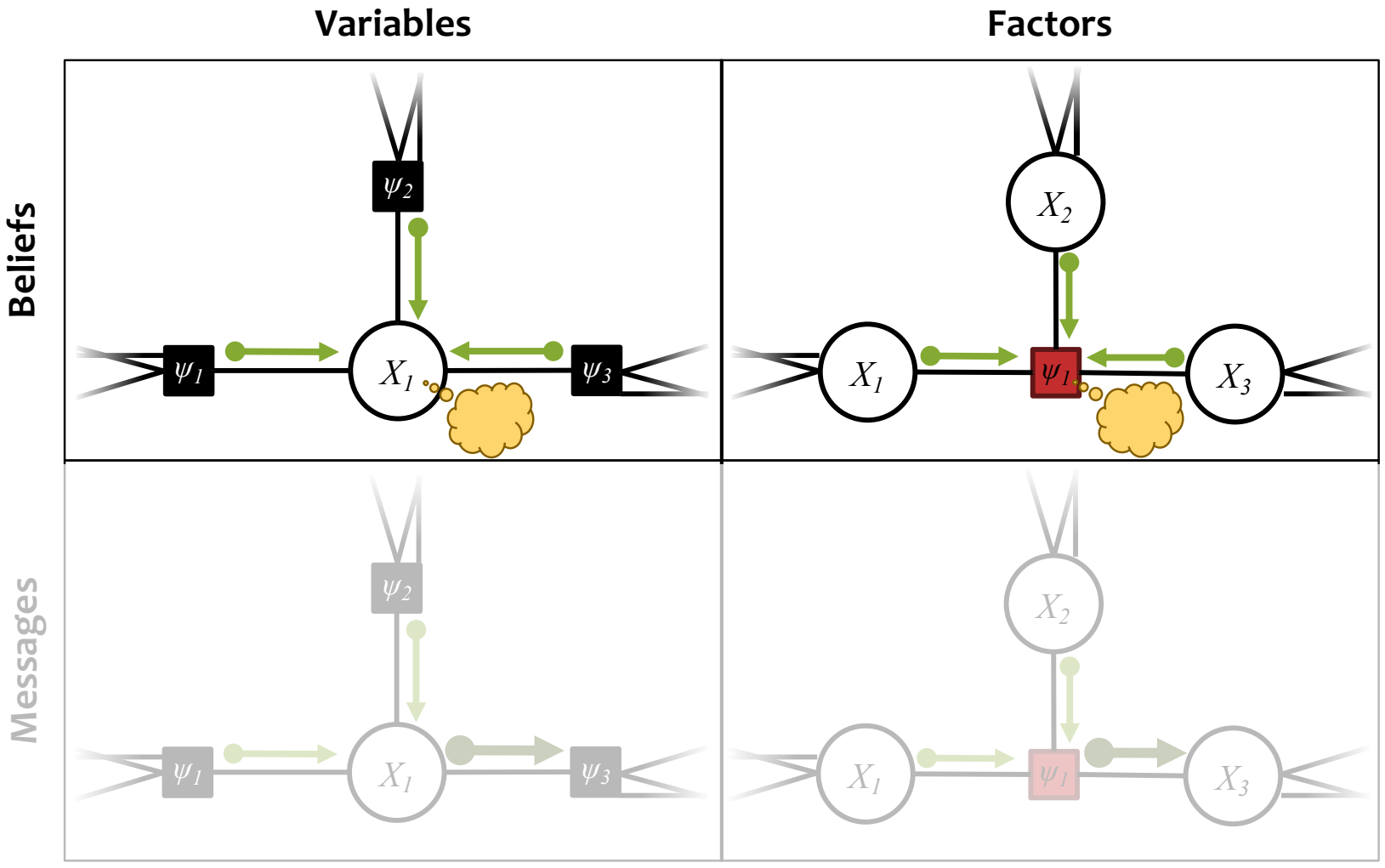
1. Compute the beliefs (unnormalized marginals).

$$b_i(x_i) = \prod_{\alpha\in\mathcal{N}(i)} \mu_{\alpha\to i}(x_i) \qquad b_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}}) = \psi_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}}) \prod_{i\in\mathcal{N}(\alpha)} \mu_{i\to\alpha}(\boldsymbol{x}_{\boldsymbol{\alpha}}[i])$$

2. Normalize beliefs and return the **exact** marginals.

$$p_i(x_i) \propto b_i(x_i) \qquad p_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}}) \propto b_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}})$$

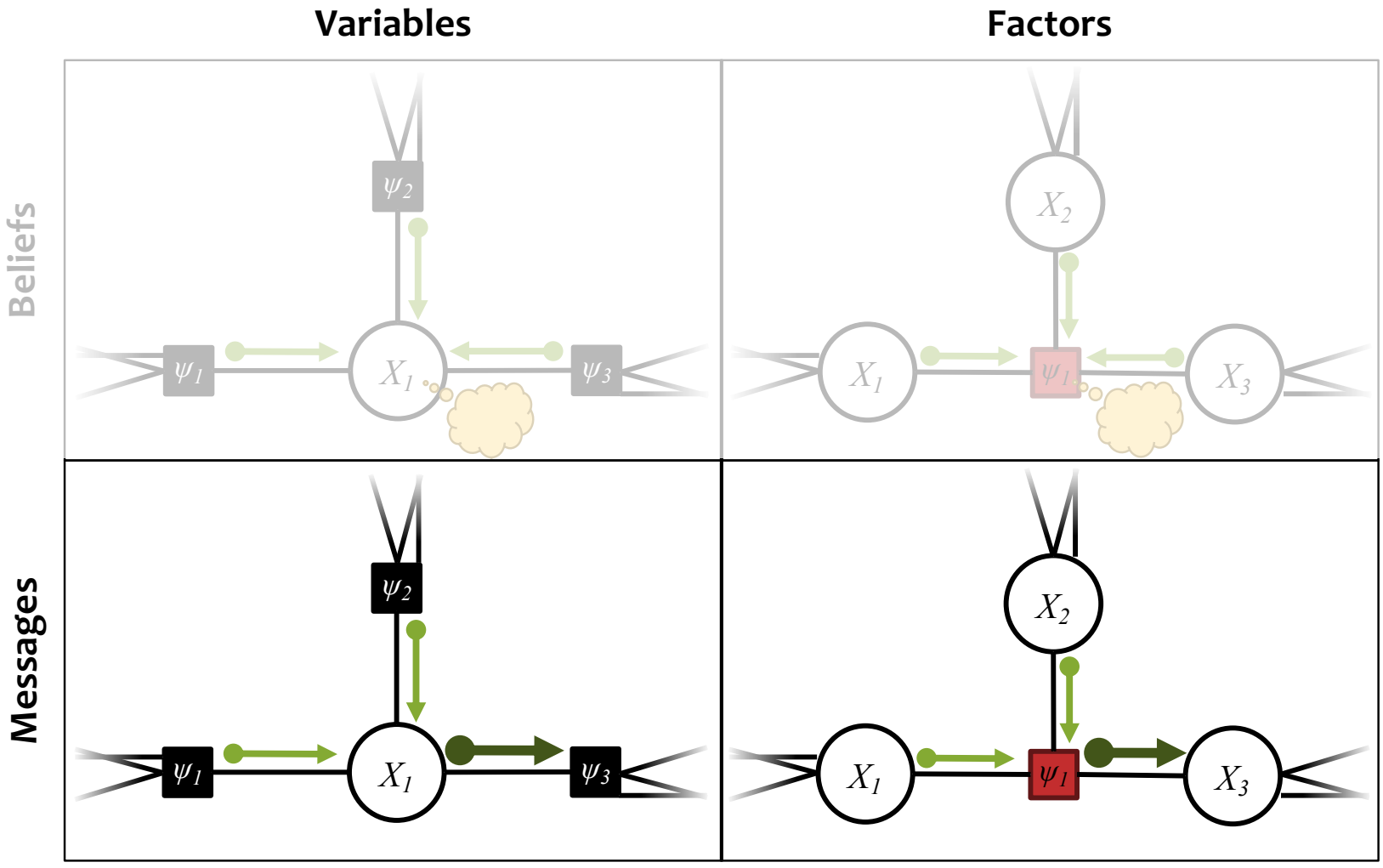
Sum-Product Belief Propagation



$$b_i(x_i) = \prod_{\alpha \in \mathcal{N}(i)} \mu_{\alpha \to i}(x_i)$$

$$b_\alpha(\boldsymbol{x}_\alpha) = \psi_\alpha(\boldsymbol{x}_\alpha) \prod_{i \in \mathcal{N}(\alpha)} \mu_{i \to \alpha}(\boldsymbol{x}_\alpha[i])$$

Sum-Product Belief Propagation



$$\mu_{i\to\alpha}(x_i) = \prod_{\alpha\in\mathcal{N}(i)\setminus\alpha} \mu_{\alpha\to i}(x_i)$$

$$\mu_{\alpha\to i}(x_i) = \sum_{\boldsymbol{x}_{\boldsymbol{\alpha}}:\boldsymbol{x}_{\boldsymbol{\alpha}}[i]=x_i} \psi_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}}) \prod_{j\in\mathcal{N}(\alpha)\setminus i} \mu_{j\to\alpha}(\boldsymbol{x}_{\boldsymbol{\alpha}}[i])$$

Sum-Product Belief Propagation

Input: a factor graph with no cycles

Output: exact marginals for each variable and factor

Algorithm:

1. Initialize the messages to the uniform distribution.

$$\mu_{i\to\alpha}(x_i) = 1 \qquad \mu_{\alpha\to i}(x_i) = 1$$

1. Choose a root node.
2. Send messages from the **leaves** to the **root**.
   Send messages from the **root** to the **leaves**.

$$\mu_{i\to\alpha}(x_i) = \prod_{\alpha\in\mathcal{N}(i)\setminus\alpha} \mu_{\alpha\to i}(x_i) \qquad \mu_{\alpha\to i}(x_i) = \sum_{\boldsymbol{x}_{\boldsymbol{\alpha}}:\boldsymbol{x}_{\boldsymbol{\alpha}}[i]=x_i} \psi_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}}) \prod_{j\in\mathcal{N}(\alpha)\setminus i} \mu_{j\to\alpha}(\boldsymbol{x}_{\boldsymbol{\alpha}}[i])$$

1. Compute the beliefs (unnormalized marginals).

$$b_i(x_i) = \prod_{\alpha\in\mathcal{N}(i)} \mu_{\alpha\to i}(x_i) \qquad b_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}}) = \psi_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}}) \prod_{i\in\mathcal{N}(\alpha)} \mu_{i\to\alpha}(\boldsymbol{x}_{\boldsymbol{\alpha}}[i])$$

2. Normalize beliefs and return the **exact** marginals.

$$p_i(x_i) \propto b_i(x_i) \qquad p_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}}) \propto b_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}})$$

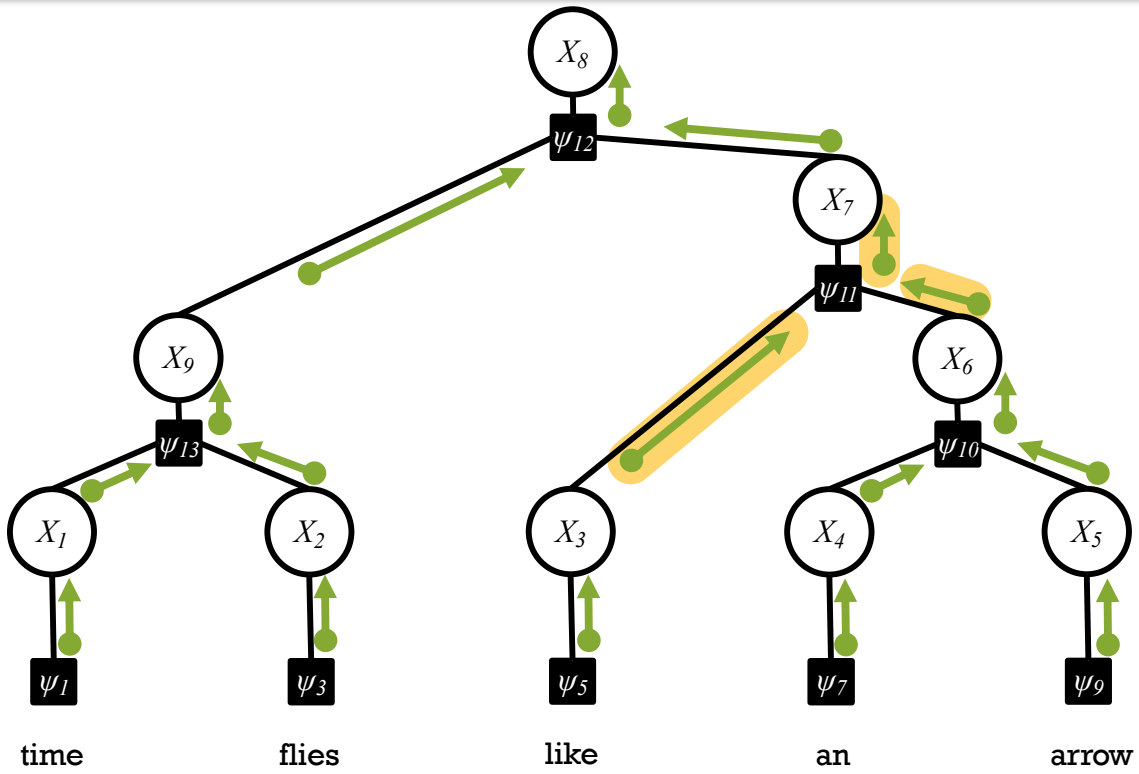
BP AS DYNAMIC PROGRAMMING

(Acyclic) Belief Propagation

In a factor graph with no cycles:

1. Pick any node to serve as the root.
2. Send messages from the **leaves** to the **root**.
3. Send messages from the **root** to the **leaves**.

A node computes an outgoing message along an edge only after it has received incoming messages along all its other edges.

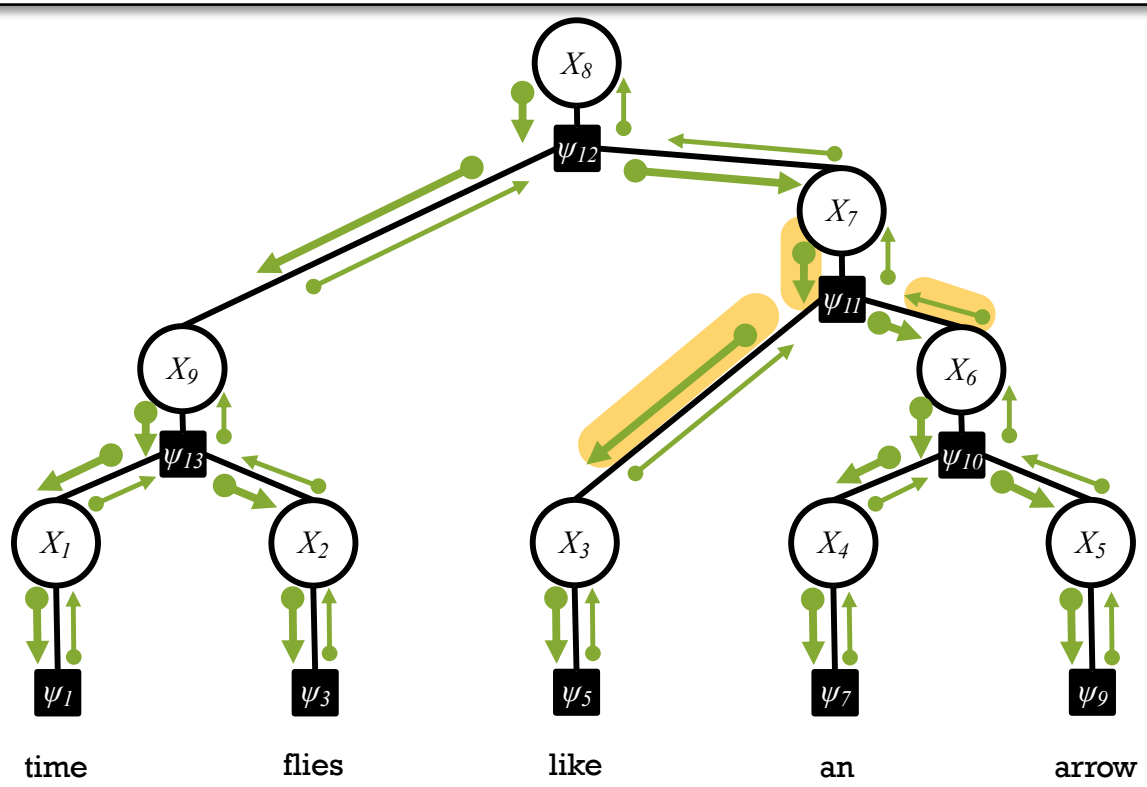


(Acyclic) Belief Propagation

In a factor graph with no cycles:

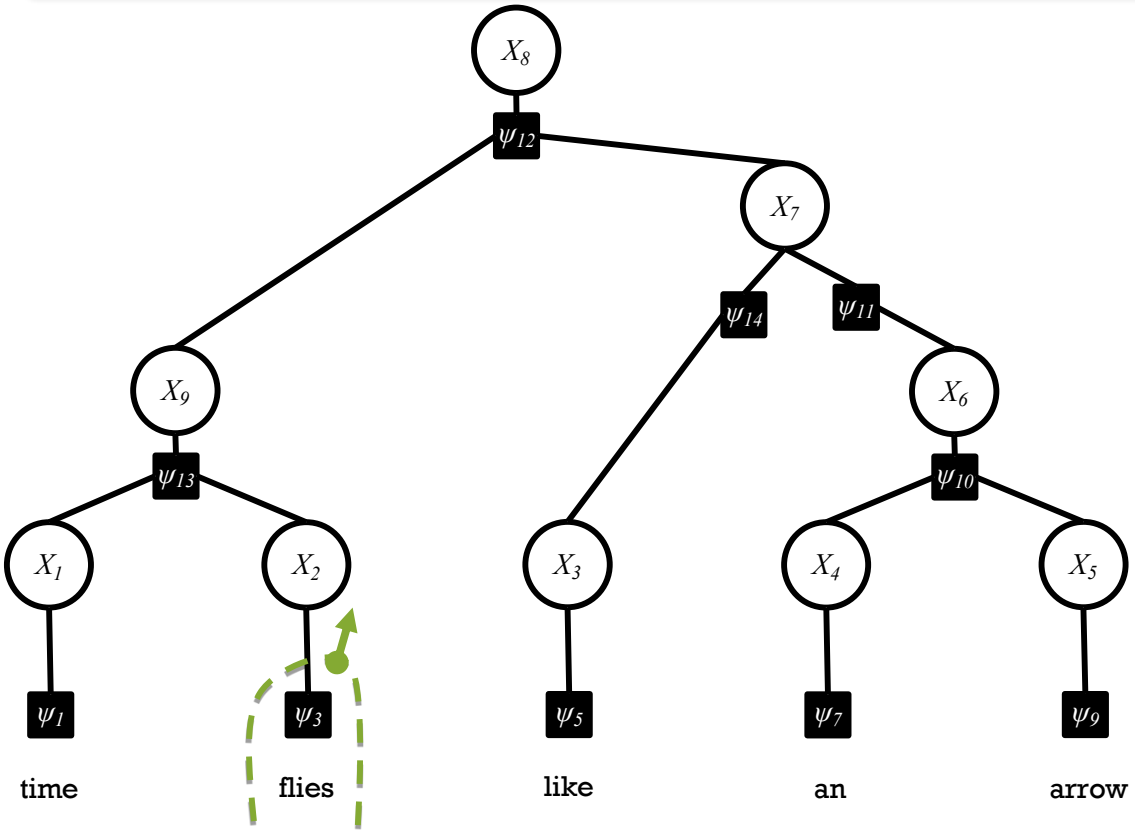
1. Pick any node to serve as the root.
2. Send messages from the **leaves** to the **root**.
3. Send messages from the **root** to the **leaves**.

A node computes an outgoing message along an edge only after it has received incoming messages along all its other edges.



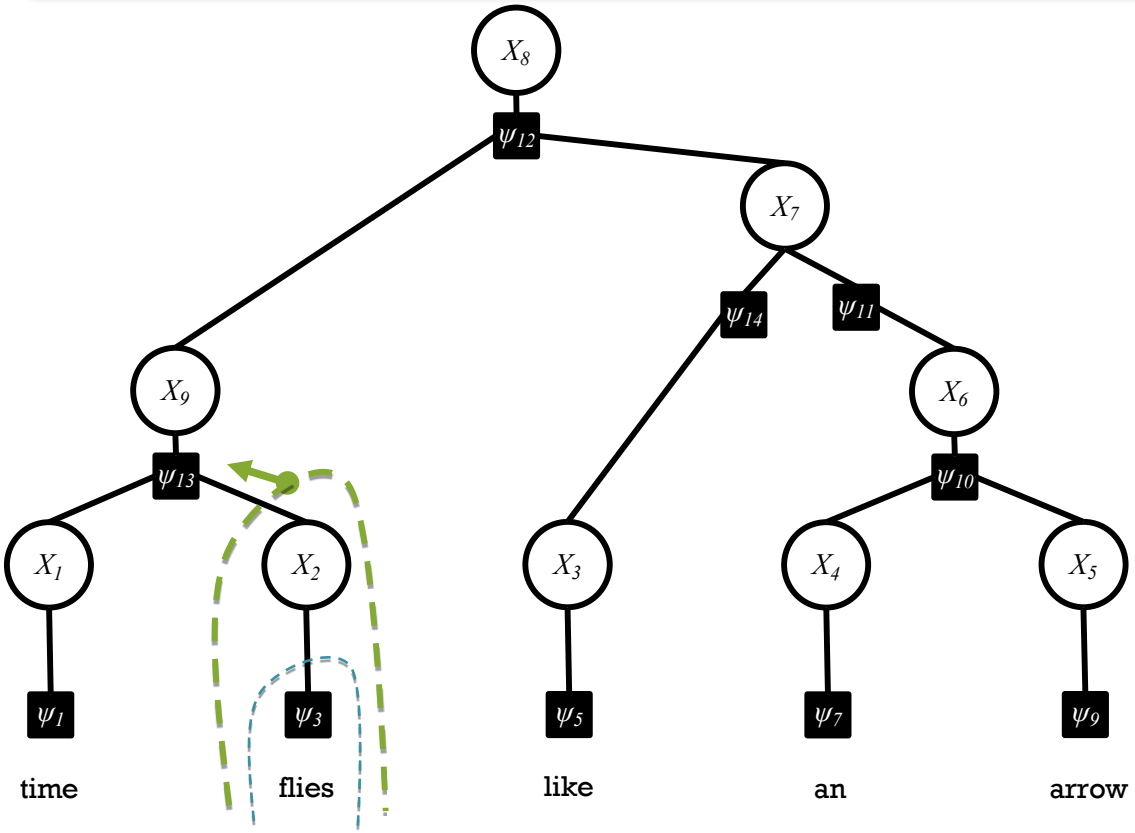
Acyclic BP as Dynamic Programming

- If you want the **marginal** $p_i(x_i)$ where X_i has degree k, you can think of that summation as a **product of k marginals** computed on smaller subgraphs.
- Each subgraph is obtained by **cutting** some edge of the tree.
- The message-passing algorithm uses **dynamic programming** to compute the marginals on all such subgraphs, working from **smaller to bigger**. So you can compute all the marginals.



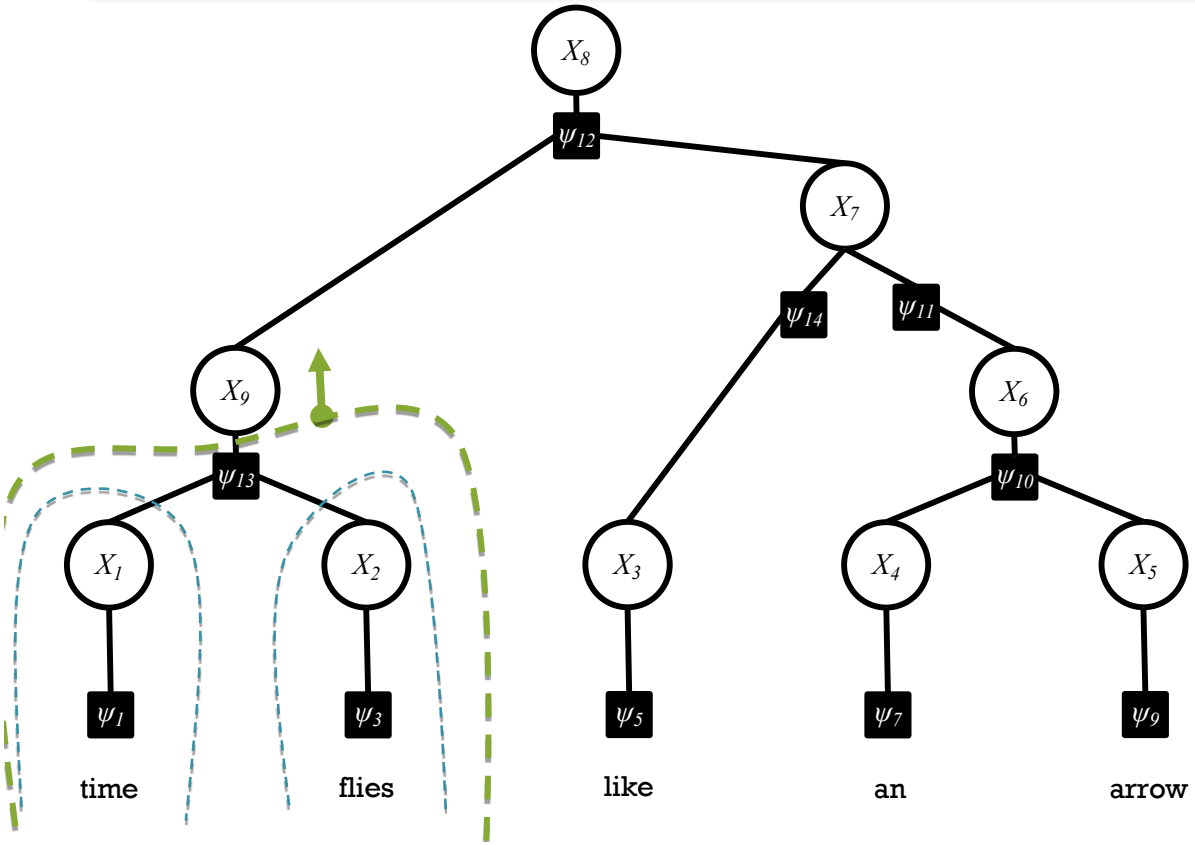
Acyclic BP as Dynamic Programming

- If you want the **marginal** $p_i(x_i)$ where X_i has degree k, you can think of that summation as a **product of k marginals** computed on smaller subgraphs.
- Each subgraph is obtained by **cutting** some edge of the tree.
- The message-passing algorithm uses **dynamic programming** to compute the marginals on all such subgraphs, working from **smaller to bigger**. So you can compute all the marginals.



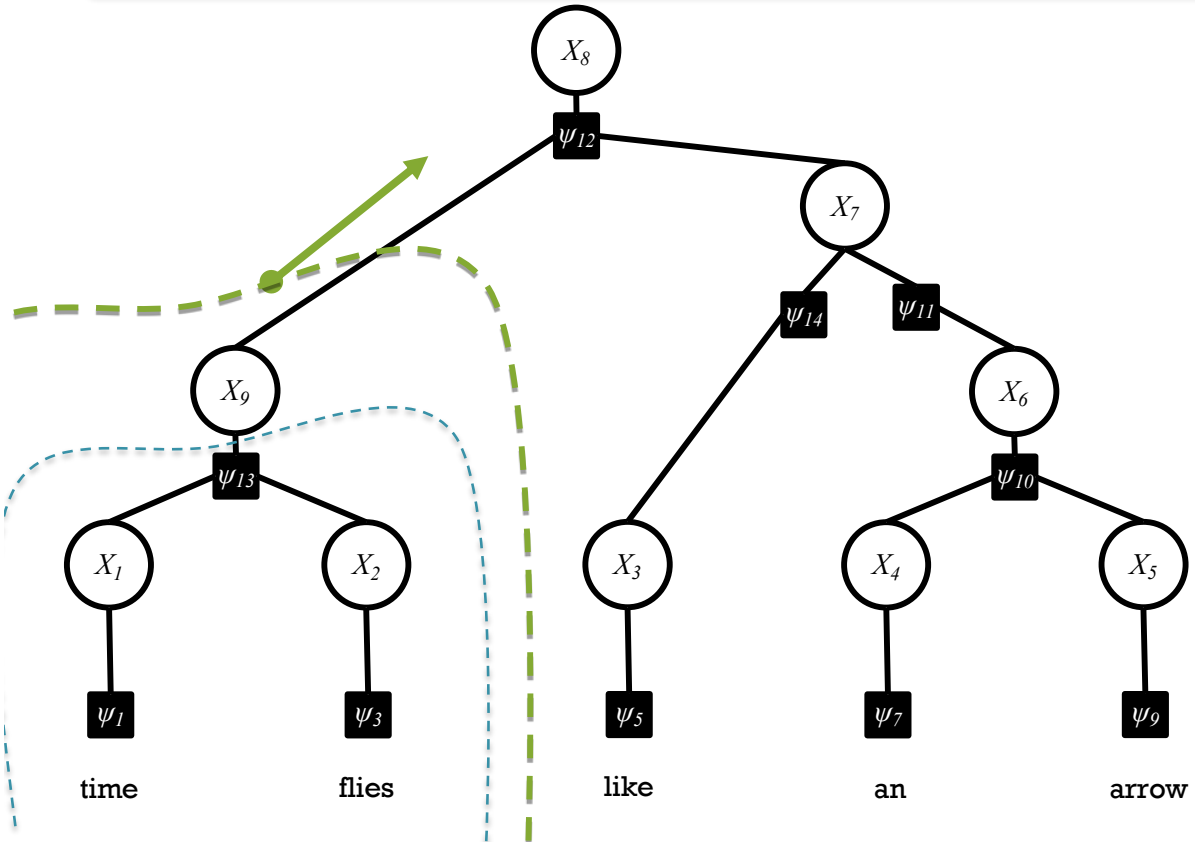
Acyclic BP as Dynamic Programming

- If you want the **marginal** $p_i(x_i)$ where X_i has degree k, you can think of that summation as a **product of k marginals** computed on smaller subgraphs.
- Each subgraph is obtained by **cutting** some edge of the tree.
- The message-passing algorithm uses **dynamic programming** to compute the marginals on all such subgraphs, working from **smaller to bigger**. So you can compute all the marginals.



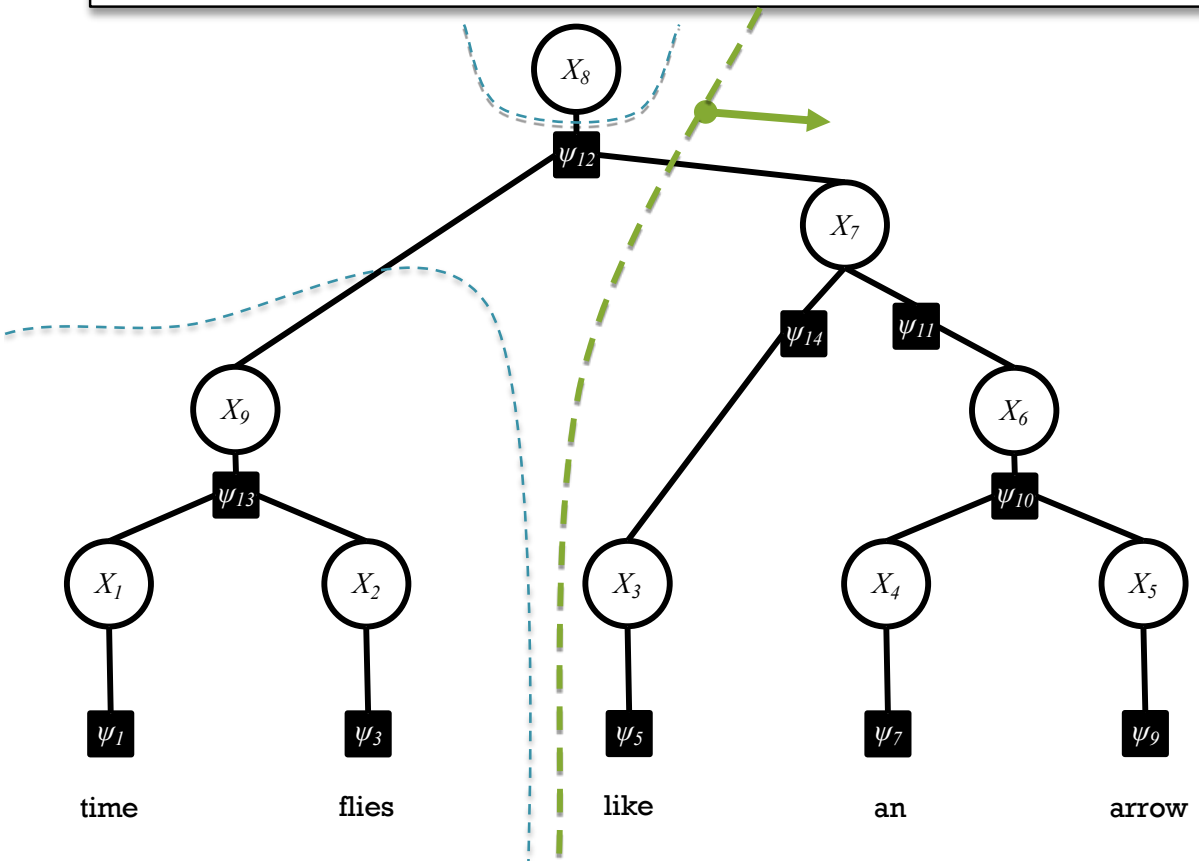
Acyclic BP as Dynamic Programming

- If you want the **marginal** $p_i(x_i)$ where X_i has degree k, you can think of that summation as a **product of k marginals** computed on smaller subgraphs.
- Each subgraph is obtained by **cutting** some edge of the tree.
- The message-passing algorithm uses **dynamic programming** to compute the marginals on all such subgraphs, working from **smaller to bigger**. So you can compute all the marginals.



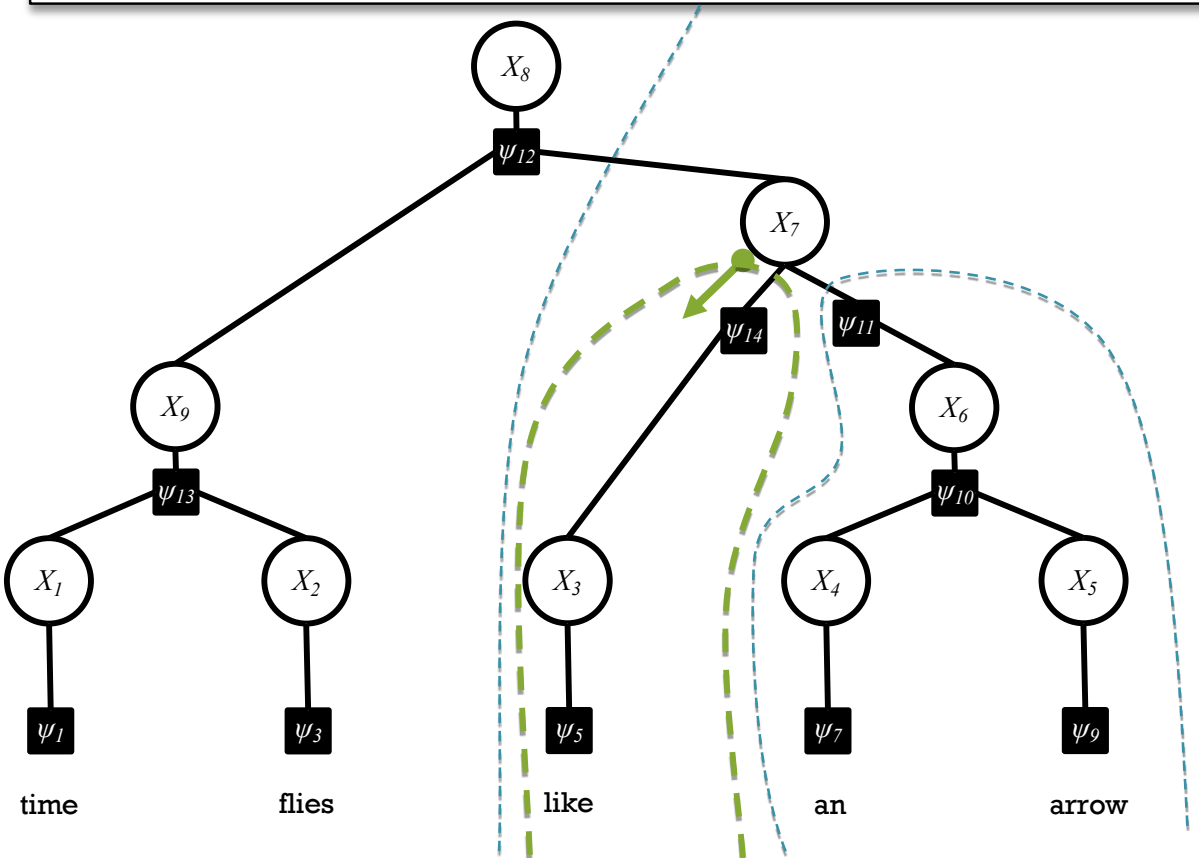
Acyclic BP as Dynamic Programming

- If you want the **marginal** $p_i(x_i)$ where X_i has degree k, you can think of that summation as a **product of k marginals** computed on smaller subgraphs.
- Each subgraph is obtained by **cutting** some edge of the tree.
- The message-passing algorithm uses **dynamic programming** to compute the marginals on all such subgraphs, working from **smaller to bigger**. So you can compute all the marginals.



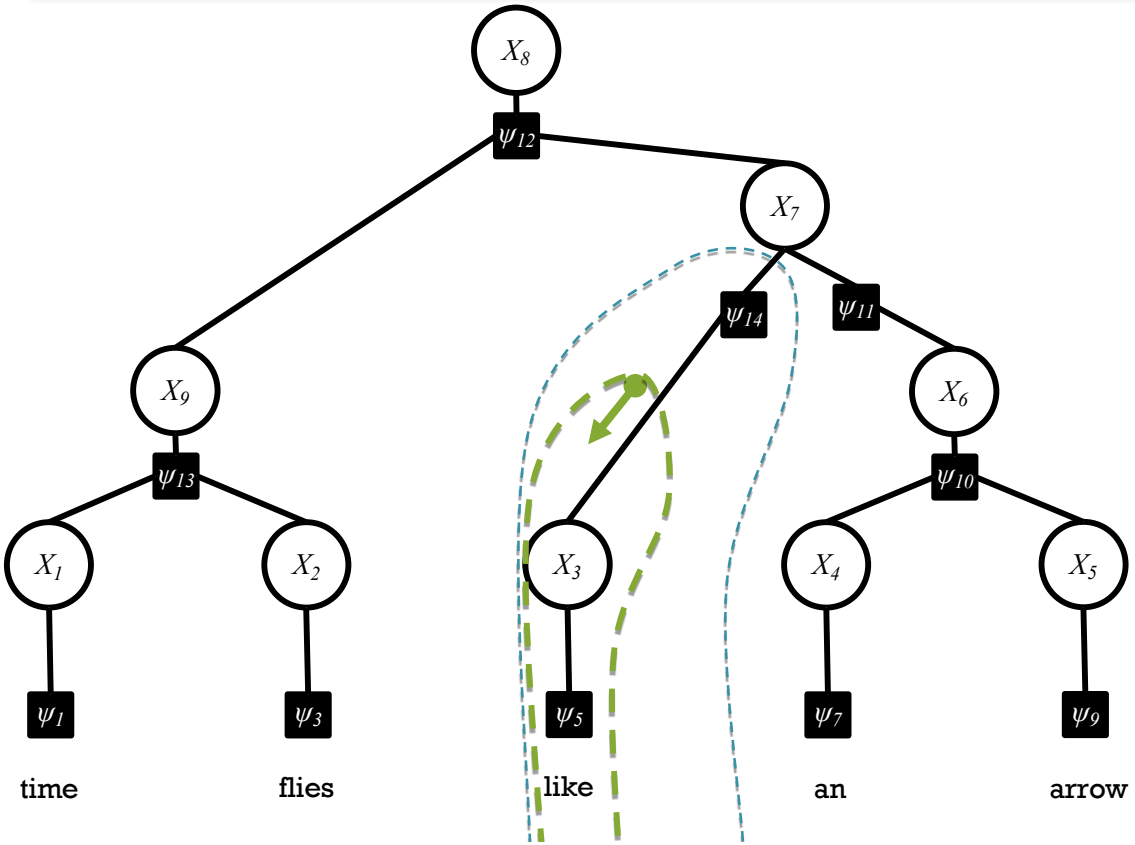
Acyclic BP as Dynamic Programming

- If you want the **marginal** $p_i(x_i)$ where X_i has degree k, you can think of that summation as a **product of k marginals** computed on smaller subgraphs.
- Each subgraph is obtained by **cutting** some edge of the tree.
- The message-passing algorithm uses **dynamic programming** to compute the marginals on all such subgraphs, working from **smaller to bigger**. So you can compute all the marginals.



Acyclic BP as Dynamic Programming

- If you want the **marginal** $p_i(x_i)$ where X_i has degree k, you can think of that summation as a **product of k marginals** computed on smaller subgraphs.
- Each subgraph is obtained by **cutting** some edge of the tree.
- The message-passing algorithm uses **dynamic programming** to compute the marginals on all such subgraphs, working from **smaller to bigger**. So you can compute all the marginals.



Sequential vs Parallel BP

- You can run BP in sequentially or in parallel
- To run it in parallel:
  - Initialize messages to 1
  - Send all messages simultaneously, and repeat until there are no changes
  - For trees, this will converge

Exact MAP inference for factor trees

MAX-PRODUCT BELIEF PROPAGATION

Max-product Belief Propagation

- **Sum-product BP** can be used to
  compute the marginals, $p_i(X_i)$
  compute the partition function, Z

- **Max-product BP** can be used to
  compute the most likely assignment,
  $X^* = \text{argmax}_X \, p(X)$

Max-product Belief Propagation

- Change the sum to a max:

$$\mu_{i\to\alpha}(x_i) = \prod_{\alpha\in\mathcal{N}(i)\setminus\alpha} \mu_{\alpha\to i}(x_i)$$

$$\mu_{\alpha\to i}(x_i) = \sum_{\boldsymbol{x_\alpha}:\boldsymbol{x_\alpha}[i]=x_i} \psi_\alpha(\boldsymbol{x_\alpha}) \prod_{j\in\mathcal{N}(\alpha)\setminus i} \mu_{j\to\alpha}(\boldsymbol{x_\alpha}[i])$$

- **Max-product BP** computes **max-marginals**
  - The max-marginal $b_i(x_i)$ is the (unnormalized) probability of the MAP assignment under the constraint $X_i = x_i$.
  - For an acyclic graph, the MAP assignment (assuming there are no ties) is given by:

$$x_i^* = \arg\max_{x_i} b_i(x_i)$$

Max-product Belief Propagation

- Change the sum to a max:

$$\mu_{i\to\alpha}(x_i) = \prod_{\alpha\in\mathcal{N}(i)\setminus\alpha} \mu_{\alpha\to i}(x_i)$$

$$\mu_{\alpha\to i}(x_i) = \max_{\boldsymbol{x}_{\boldsymbol{\alpha}}:\boldsymbol{x}_{\boldsymbol{\alpha}}[i]=x_i} \psi_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}}) \prod_{j\in\mathcal{N}(\alpha)\setminus i} \mu_{j\to\alpha}(\boldsymbol{x}_{\boldsymbol{\alpha}}[i])$$

- **Max-product BP** computes **max-marginals**
  - The max-marginal $b_i(x_i)$ is the (unnormalized) probability of the MAP assignment under the constraint $X_i = x_i$.
  - For an acyclic graph, the MAP assignment (assuming there are no ties) is given by:

$$x_i^* = \arg\max_{x_i} b_i(x_i)$$

Semirings

- Sum-product +/* and max-product max/* are commutative semirings
- We can run BP with any such commutative semiring

$$\mu_{i \to \alpha}(x_i) = \prod_{\alpha \in \mathcal{N}(i) \setminus \alpha} \mu_{\alpha \to i}(x_i)$$

$$\mu_{\alpha \to i}(x_i) = \sum_{\boldsymbol{x}_{\boldsymbol{\alpha}} : \boldsymbol{x}_{\boldsymbol{\alpha}}[i] = x_i} \psi_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}}) \prod_{j \in \mathcal{N}(\alpha) \setminus i} \mu_{j \to \alpha}(\boldsymbol{x}_{\boldsymbol{\alpha}}[i])$$

- In practice, multiplying many small numbers together can yield underflow
  - instead of using +/*, we use log-add/+
  - Instead of using max/*, we use max/+

Deterministic Annealing

Motivation: Smoothly transition from sum-product to max-product

1. Incorporate inverse temperature parameter into each factor:

Annealed Joint Distribution

$$p(\boldsymbol{x}) = \frac{1}{Z} \prod_{\alpha} \psi_{\alpha}(\boldsymbol{x}_{\boldsymbol{\alpha}})^{\frac{1}{T}}$$

1. Send messages as usual for sum-product BP
2. Anneal T from 1 to 0:

$T = 1$	Sum-product
$T \rightarrow 0$	Max-product

3. Take resulting beliefs to power T

Summary

1. **Factor Graphs**
   - Alternative representation of directed / undirected graphical models
   - Make the cliques of an undirected GM explicit
2. **Variable Elimination**
   - Simple and general approach to exact inference
   - Just a matter of being clever when computing sum-products
3. **Sum-product Belief Propagation**
   - Computes all the marginals and the partition function in only twice the work of Variable Elimination
4. **Max-product Belief Propagation**
   - Identical to sum-product BP, but changes the semiring
   - Computes: max-marginals, probability of MAP assignment, and (with backpointers) the MAP assignment itself.

Loopy Belief Propagation (Loopy BP)

What about graphs with loops?

- You can run the algorithm anyways
- It may not converge
- It usually gives an ok result (it is an **approximate algorithm**)

Semi-CRFs

Segmentation models (Semi-CRFs)

i	1	2	3	4	5	6	7	8	9
x	I	went	skiing	with	Fernando	Pereira	in	British	Columbia
y	O	O	O	O	I	I	O	I	I

$f^K(i, \mathbf{x}, \mathbf{y}_i, \mathbf{y}_{i-1})$

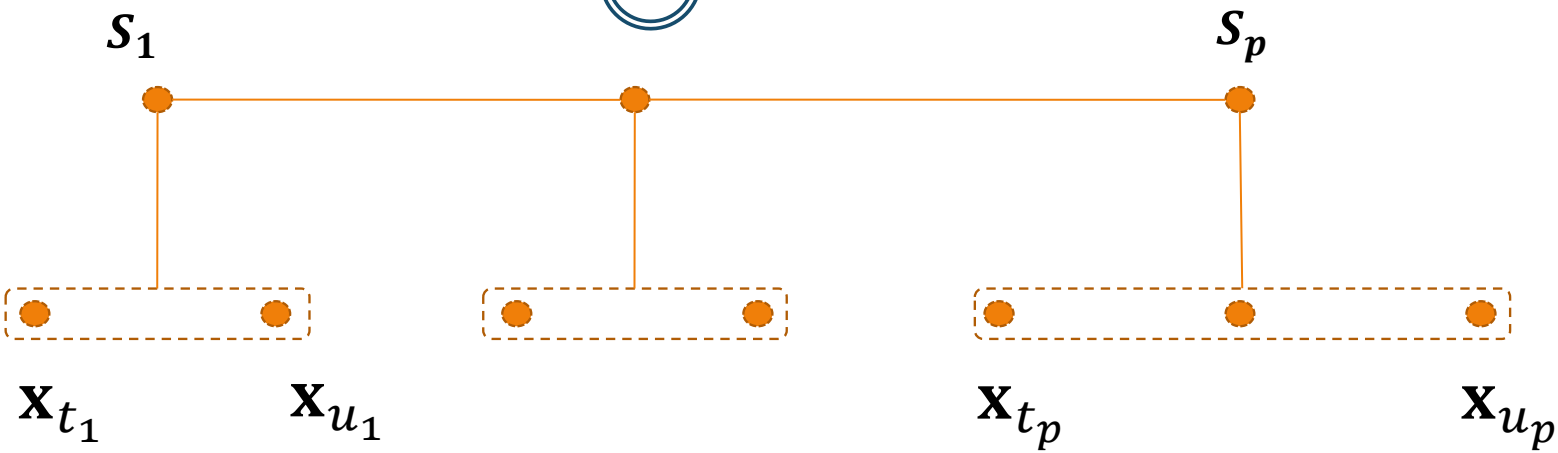
Features describe the single **word**

t,u	$t_1=u_1=1$	$t_2=u_2=2$	$t_3=u_3=3$	$t_4=u_4=4$	$t_5=5, u_5=6$	$t_6=u_6=7$	$t_7=8, u_7=9$
x	I	went	skiing	with	Fernando Pereira	in	British Columbia
y	O	O	O	O	I	O	I

$g^K(\mathrm{y}_j, \mathrm{y}_{j-1}, \mathbf{x}, t_j, u_j)$

Features describe the **segment** from t_j to u_j

Semi-CRF

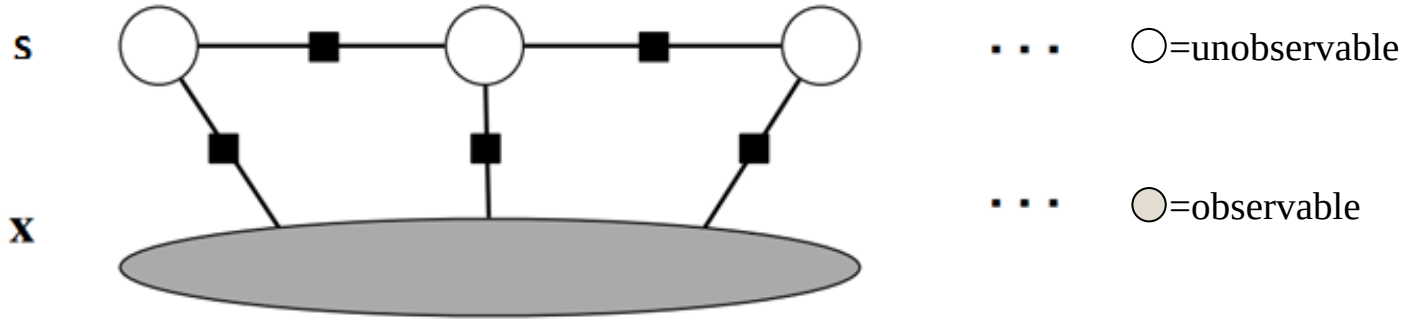


$\mathbf{s} = \langle s_1, \ldots, s_p \rangle$ denote a segmentation of $\mathbf{x}$

Segment $s_j = \langle t_j, u_j, y_j \rangle$ consists of a start position t_j, an end position u_j, and a label y_j

$1 \leq t_j \leq u_j \leq |s|$ and $t_{j+1} = u_j + 1$

Semi-CRF



$$g^K(j, \mathbf{x}, \mathbf{s}) = g'^K(\mathrm{y}_j, \mathrm{y}_{j-1}, \mathbf{x}, t_j, u_j)$$

$$\Pr(\mathbf{s}|\mathbf{x}, \mathbf{W}) = \frac{1}{Z(\mathbf{x})} e^{\mathbf{W}^{\mathrm{T}}\mathbf{G}(\mathbf{x},\mathbf{s})}$$

$$\mathbf{G}(\mathbf{x}, \mathbf{s}) = \sum_{i=1}^{|\mathbf{x}|} \mathbf{g}(\mathrm{i}, \mathbf{x}, \mathbf{s}) \qquad Z(\mathbf{x}) = \sum_{s'} e^{\mathbf{W}^{\mathrm{T}}\mathbf{G}(\mathbf{x},\mathbf{s})}$$

g is a vector of segment level feature functions.

MAP Inference Semi-CRF

$$\mathbf{S}^* = \underset{\mathbf{s}}{\operatorname{argmax}}\, P(\mathbf{s}|\mathbf{x}, \mathbf{w})$$

$$\mathbf{S}^* = \underset{\mathbf{s}}{\operatorname{argmax}}\, \mathbf{W}^{\mathrm{T}}\mathbf{G}(\mathbf{x}, \mathbf{s})$$

$$\mathbf{S}^* = \underset{\mathbf{s}}{\operatorname{argmax}}\, \mathbf{W}^{\mathrm{T}} \sum_{j}^{|\mathbf{s}|} \mathbf{g}\left(\mathrm{y}_j, \mathrm{y}_{j-1}, \mathbf{x}, t_j, u_j\right)$$

g is a vector of segment level feature functions.

Viterbi algorithm for Semi-CRF

$$\max_{\mathbf{s}} \mathbf{W}^{\mathrm{T}} \sum_{\boldsymbol{j=1}}^{|\mathbf{s}|} \mathbf{g}(\mathrm{y}_j, \mathrm{y}_{j-1}, \mathbf{x}, t_j, u_j)$$

- L be an upper bound on segment length
- $\mathbf{s_{i:y}}$ denote set of all partial segmentation starting from 1 to i, such that the last segment has the label y and ending position i.

$$V(i,y) = \max_{\boldsymbol{y',d}} \max_{\mathbf{s'} \in \mathbf{s_{i-d:y'}}} \mathbf{W}^{\mathrm{T}} \sum_{\boldsymbol{j=1}}^{|\mathbf{s'}|} \mathbf{g}(\mathrm{y}_j, \mathrm{y}_{j-1}, \mathbf{x}, t_j, u_j) + \mathbf{W}^{\mathrm{T}} \mathbf{g}(y, y', \mathbf{x}, i-d, i)$$

Viterbi algorithm for Semi-CRF

$$V(i,y) = \max_{y',d} \max_{s' \in s_{i-d:y'}} \mathbf{W}^{\mathrm{T}} \sum_{j=1}^{|s'|} \mathbf{g}(y_j, y_{j-1}, \mathbf{x}, t_j, u_j) + \max_{y',d} \mathbf{W}^{\mathrm{T}} \mathbf{g}(y, y', \mathbf{x}, i-d, i)$$

$$V(i-d,y') = \max_{s' \in s_{i-d:y'}} \mathbf{W}^{\mathrm{T}} \sum_{j=1}^{|s'|} \mathbf{g}(y_j, y_{j-1}, \mathbf{x}, t_j, u_j)$$

$$V(i,y) = \max_{y',d} V(i-d,y') + \mathbf{W}^{\mathrm{T}} \mathbf{g}(y, y', \mathbf{x}, i-d, i)$$

Viterbi algorithm for Semi-CRF

$$V(i,y) = \begin{cases} \max\limits_{y',d=1,..L} V(i-d,y') + \mathbf{W}^{\mathrm{T}}\mathbf{g}(y,y',\mathbf{x},i-d,i) & \text{If i >o} \\ 0 & \text{If i = o} \\ -\infty & \text{If i<o} \end{cases}$$

The optimal label sequence corresponds to path traced by $\max\limits_{y} V(|x|,y)$.

Semi-Markov CRFs *vs* conventional CRFs

Since conventional CRFs need not maximize over possible segment lengths d, inference for semi-CRFs is more expensive. However additional cost is only linear in L.

Semi-CRFs are more expressive power.

A major advantage of semi-CRFs is that they allow features which measure properties of segments, rather than individual elements.

Semi-Markov CRFs *vs* Higher order CRFs

Semi-CRFs are no more expressive than order-L CRFs.

For order-L CRFs, however the additional computational cost is exponential in L.

Semi-CRFs only consider sequences in which the same label is assigned to all L positions, rather than all $|Y|^L$ length-L sequences. This is a useful restriction, as it leads to faster inference.

Parameter Learning: Semi-CRF

- Given the training data, $\{(\mathbf{x}_l, \boldsymbol{s}_l)\}_{l=1}^{N}$ we wish to learn parameters of the model. We express the log-likelihood over the training sequences as

$$L(W) = \sum_l \log P(\boldsymbol{s}_l|\mathbf{x}_l, \mathbf{W}) = \sum_l (\mathbf{W}^{\mathrm{T}}\mathbf{G}(\mathbf{x}_l, \boldsymbol{s}_l) - \log \mathrm{Z}_{\mathbf{W}}(\mathbf{x}_l))$$

$L(W)$ is concave, and can thus be maximized by gradient ascent, or one of many related methods. (Paper uses a limited-memory quasi-Newton method)

$$\nabla L(W) = \sum_l (\underbrace{\mathbf{G}(\mathbf{x}_l, \boldsymbol{s}_l)}_{\text{Observed feature count}} - \underbrace{E_{\mathrm{P}r(\mathbf{s}'|\mathbf{x}, \mathbf{W})}\, \mathbf{G}(\mathbf{x}_l, \mathbf{s}')}_{\text{Expected feature count}})$$

Parameter Learning: Semi-CRF

$$\nabla L(W) = \sum_{l} (\mathbf{G}(\mathbf{x}_l, \boldsymbol{s}_l) \quad - \quad E_{\mathrm{P}r(\mathbf{s}'|\mathbf{x}, \mathbf{W})}\, \mathbf{G}(\mathbf{x}_l, \mathbf{s}'))$$

$$\nabla L(W) = \sum_{l} \left(\mathbf{G}(\mathbf{x}_l, \boldsymbol{s}_l) \quad - \frac{\sum_{s'} \mathbf{G}(\mathbf{x}_l, s') \boldsymbol{e}^{\mathbf{W}^{\mathrm{T}} \mathbf{G}(\mathbf{x}_l, s')}}{\mathrm{Z}_{\mathbf{W}}(\mathbf{x}_l)} \right)$$

Markov property of G and a dynamic programming helps in fast computation of the expected value of the features under the current weight vector $E_{\mathrm{P}r(\mathbf{s}'|\mathbf{x}, \mathbf{W})}\, \mathbf{G}(\mathbf{x}_l, \mathbf{s}')$

$$\alpha(i, y) = \sum_{s' \in \mathbf{s_{i:y}}} \boldsymbol{e}^{\mathbf{W}^{\mathrm{T}} \mathbf{G}(\mathbf{x}_l, s')}$$

Where $\mathbf{s_{i:y}}$ denotes all segmentations from 1 to i ending at i and labeled y.

$$\mathrm{Z}_{\mathbf{W}}(\mathbf{x}) = \sum_{y} \alpha(|\mathbf{x}|, y)$$

End

We stopped here.