10-708 Probabilistic Graphical Models

Machine Learning Department
School of Computer Science
Carnegie Mellon University

Learning MRFs / CRFs

Matt Gormley
Lecture 6
Feb. 17, 2021

Q&A

Q: What is a moralized graph again?

A: Oops! My definition of d-separation confused this issue…
…let's fix that.

D-Separation

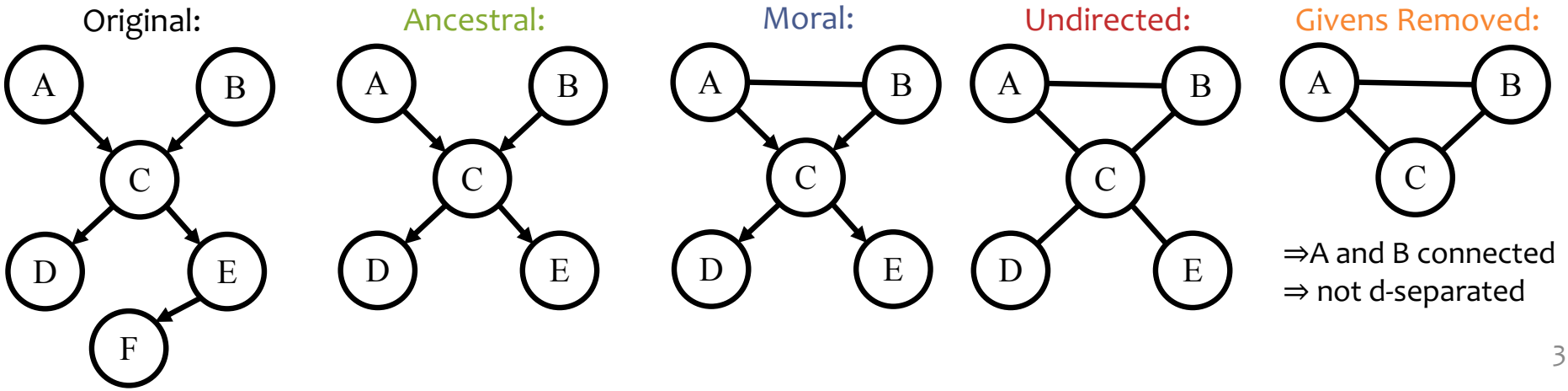
If variables X and Z are **d-separated** given a **set** of variables E
Then X and Z are **conditionally independent** given the **set** E

Definition #2:
Variables X and Z are **d-separated** given a **s**[illegible]**e** does **not** exist a path in the **undirected ancestra**[illegible]

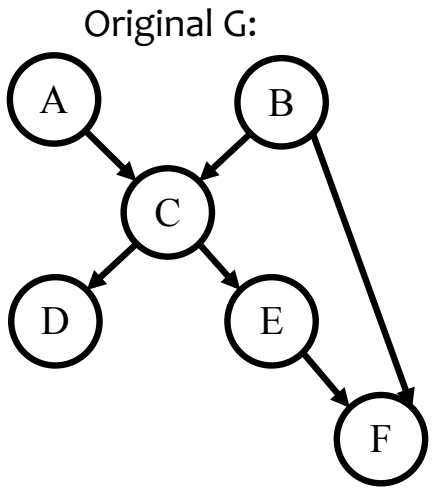
This one is the **moralized graph**

1. **Ancestral graph**: keep only X, Z, E an[illegible]
2. **Moral graph**: add undirected edge be[illegible]rents
3. **Undirected graph**: convert all directed edges to undi[illegible]ted
4. Givens Removed: delete any nodes in E

Example Query: $A \perp\!\!\!\perp B \mid \{D, E\}$



Moral Graph

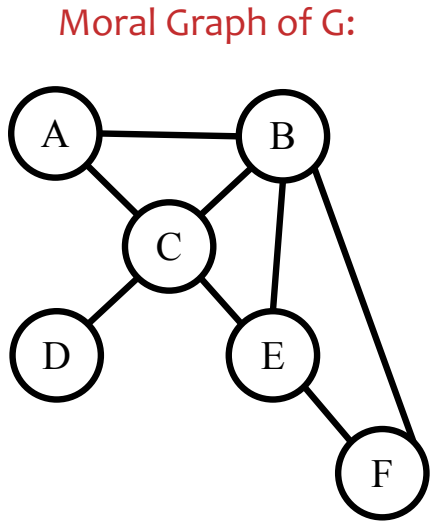


Definition #1: we create a **moralized version of a DAG** by…

1. adding an edge between each pair of nodes with a common child that don't already have an edge connecting them
2. and then **converting all edges to be undirected**

Definition #2: we create a **moralized version of a DAG** by…

1. adding the minimal set of edges so that each node in the original graph is connected to all nodes in its Markov blanket
2. and then **converting all edges to be undirected**



Q&A

Q: Are the values in the factors changing during inference?

A: No.

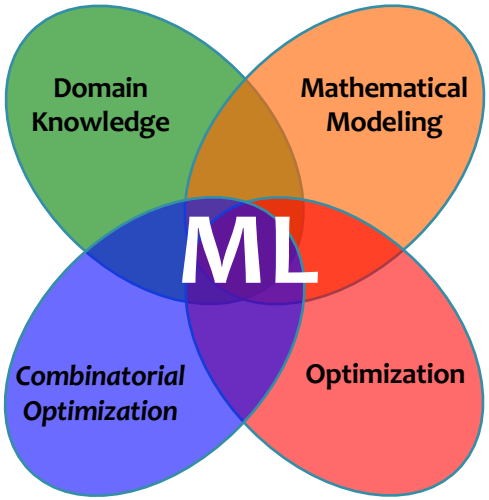
At the moment, we're assuming that someone else gave you a **pre-trained the graphical model** and gave you fixed values for each factor (i.e. CPTs for a DGM, potential functions for a UGM).

However, in today's lecture, we'll see that in order to compute the gradient of an MRF or CRF we'll need the marginals – so we'll run inference (on the current fixed set of parameters) to get those marginals.

Structured Prediction

The **data** inspires the structures we want to predict

Our **model** defines a score for each structure



It also tells us what to optimize

Inference finds {best structure, marginals, partition function} for a new observation

Learning tunes the parameters of the model

(**Inference** is usually called as a subroutine in learning)

Q&A

Q: Is the only difference between the **original HW1** and the **re-released HW1** the spacing of the solution boxes?

A: Yes.

So the real question is: why should you care about spacing?

Because we're using AI-assisted grading in Gradescope and so the position of your solutions on the page matters a lot. There's a high chance we'll make a mistake when grading if your spacing is off.

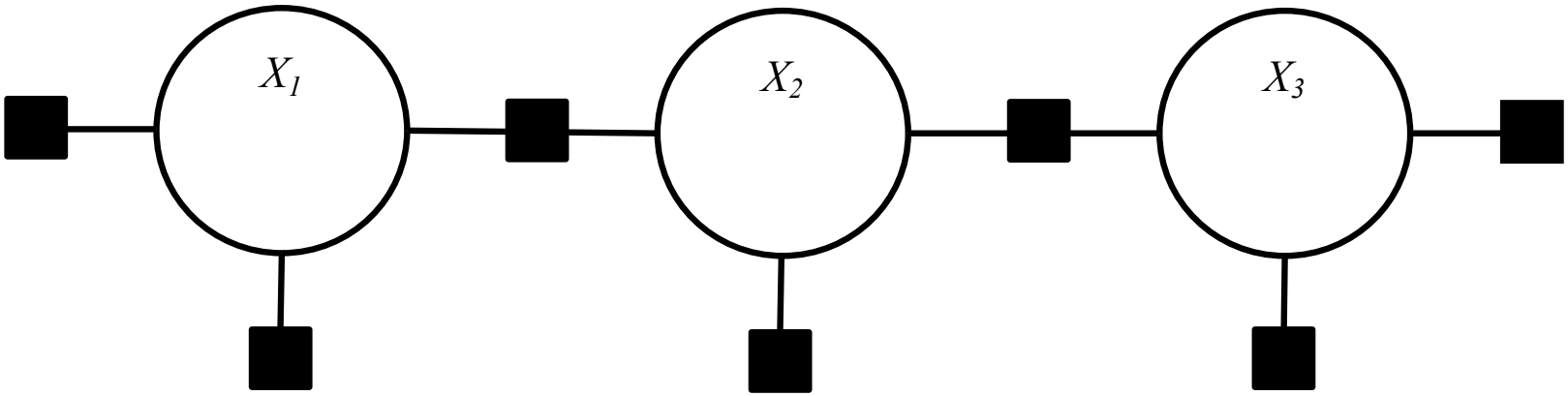
So please be sure to get the updates.

Reminders

- Homework 1: PGM Representation
  - Out: Mon, Feb. 15
  - Due: Mon, Feb. 22 at 11:59pm
- Homework 2: Exact inference and supervised learning (CRF+RNN)
  - Out: Mon, Feb. 22
  - Due: Mon, Mar. 08 at 11:59pm

FORWARD BACKWARD AS SUM-PRODUCT BP

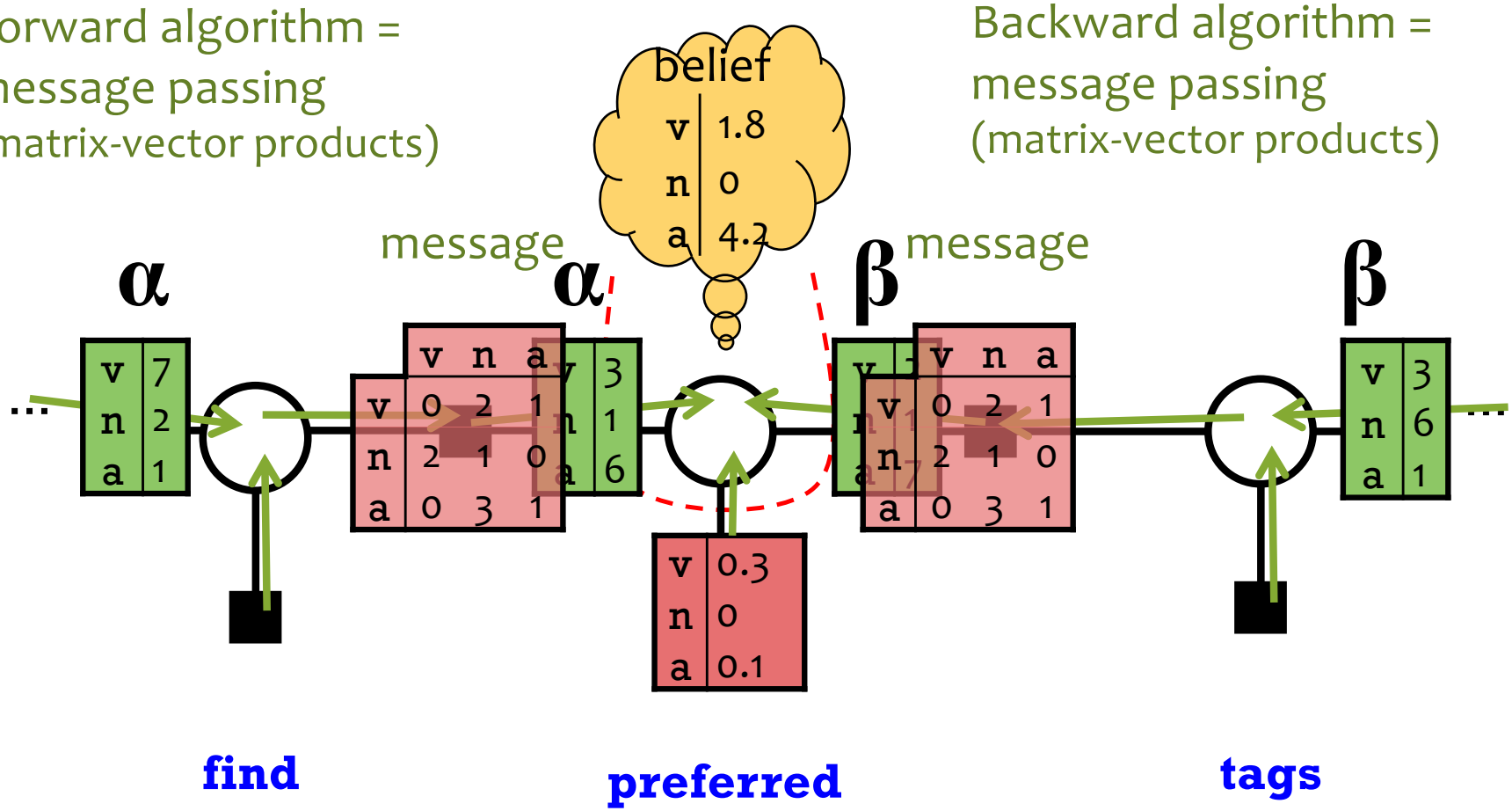
CRF Tagging Model



find preferred tags

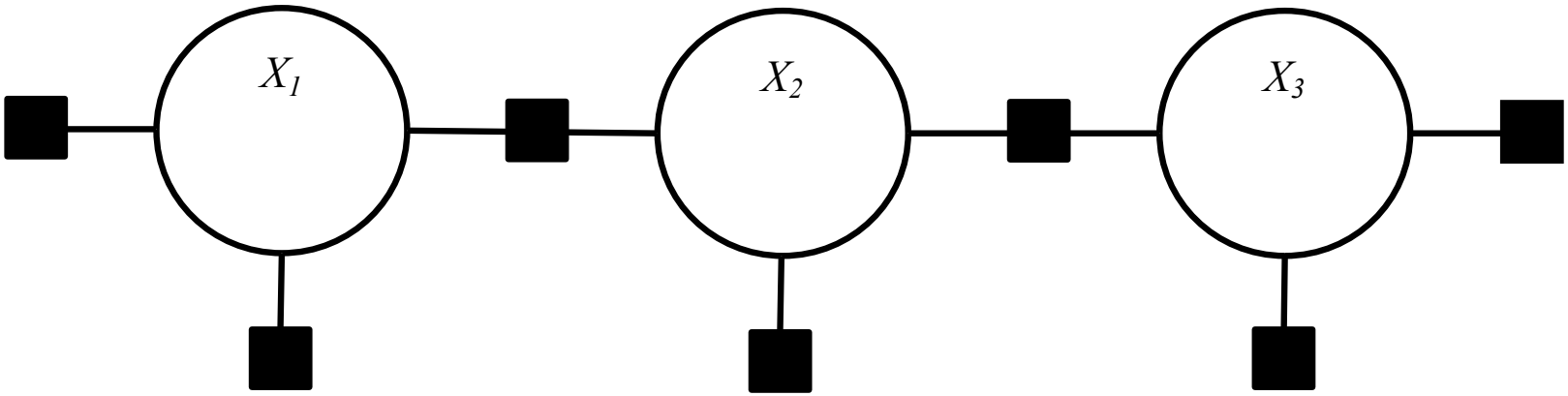
Could be verb or noun *Could be adjective or verb* *Could be noun or verb*

CRF Tagging by Belief Propagation



- Forward-backward is a message passing algorithm.
- It's the simplest case of belief propagation.

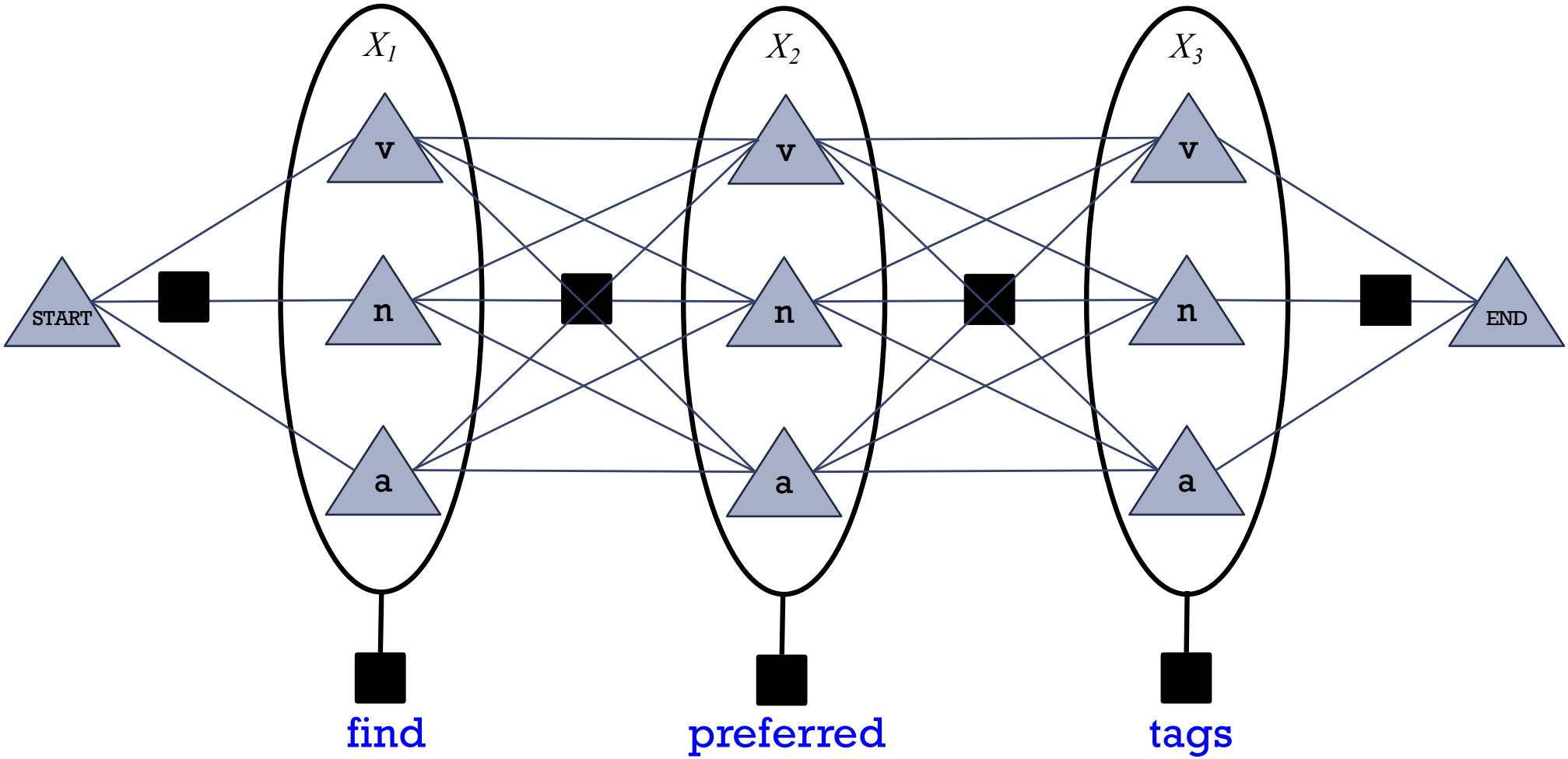
So Let's Review Forward-Backward …



find preferred tags

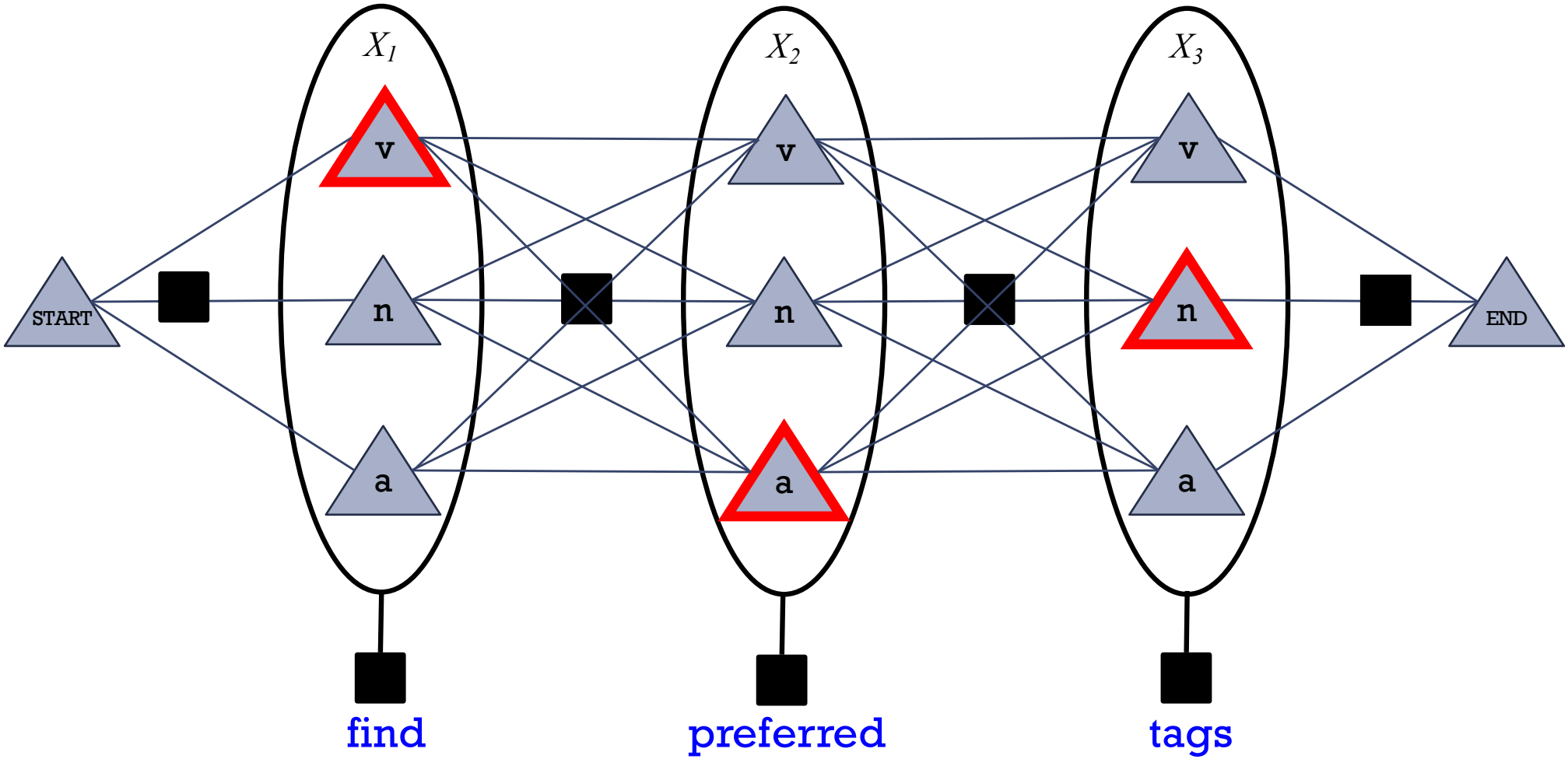
Could be verb or noun *Could be adjective or verb* *Could be noun or verb*

So Let's Review Forward-Backward …



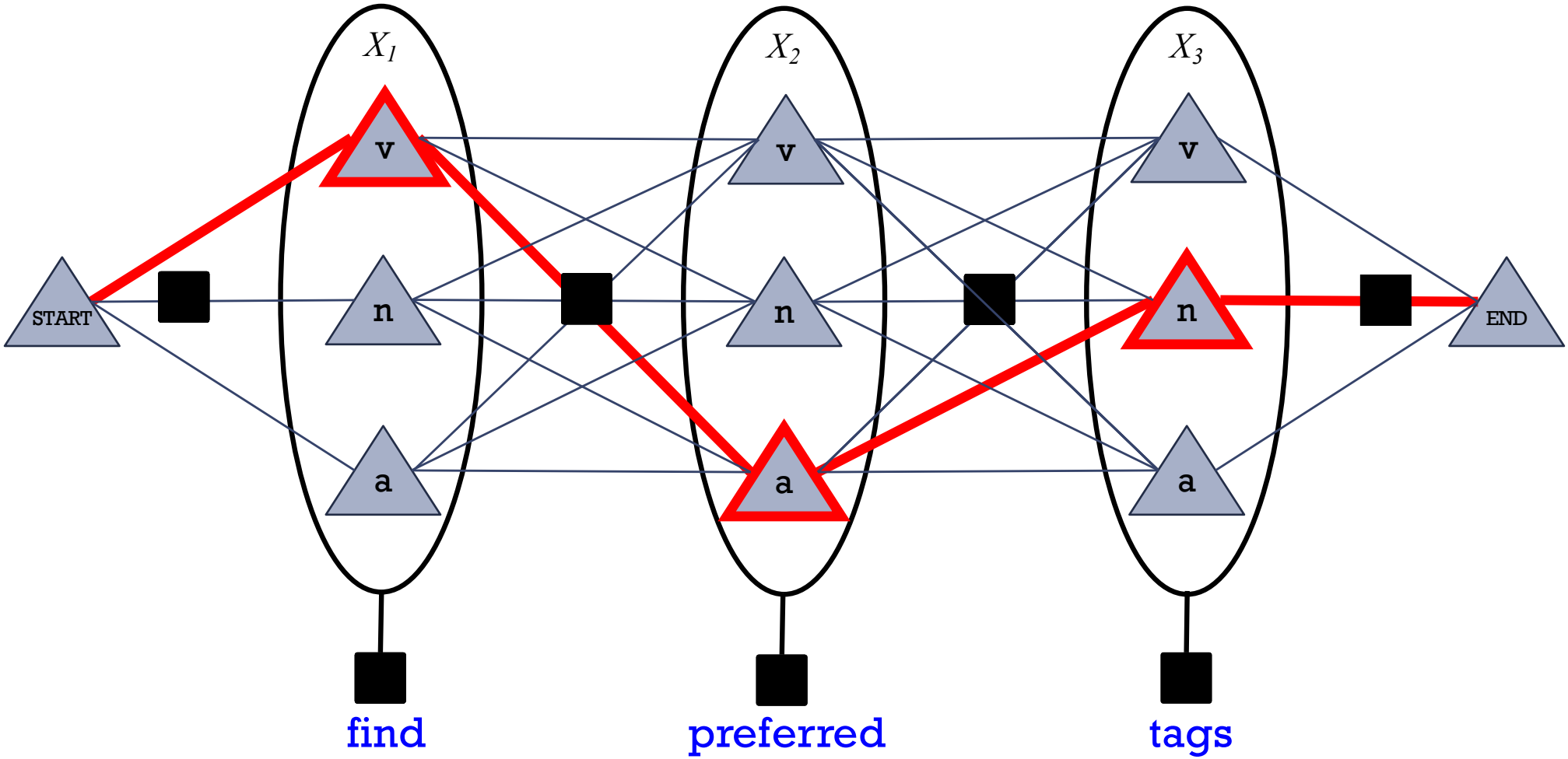
- Show the possible *values* for each variable

So Let's Review Forward-Backward …



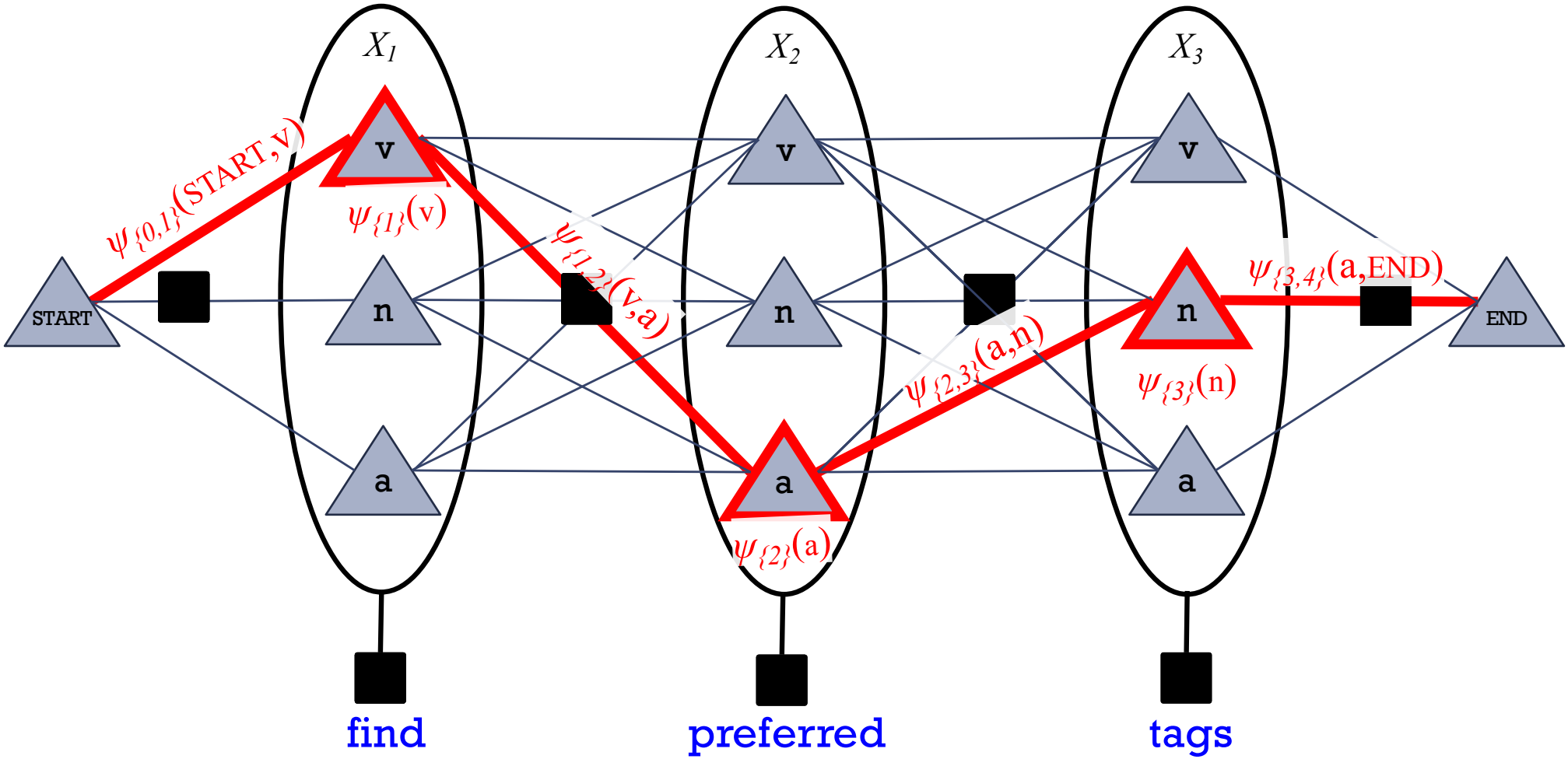
- Let's show the possible *values* for each variable
- One possible assignment

So Let's Review Forward-Backward …



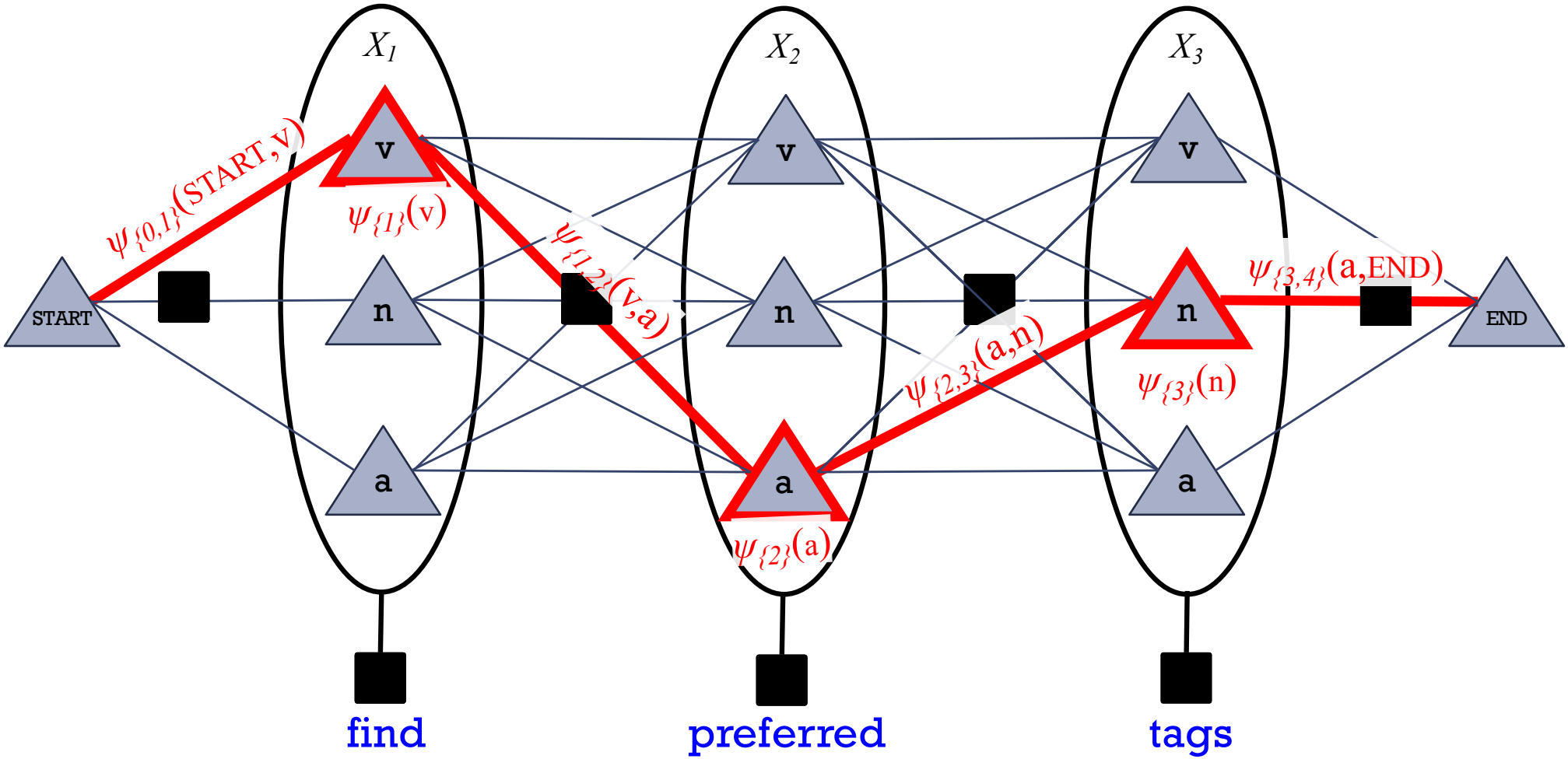
- Let's show the possible *values* for each variable
- One possible assignment
- And what the 7 factors think of it …

Viterbi Algorithm: Most Probable Assignment



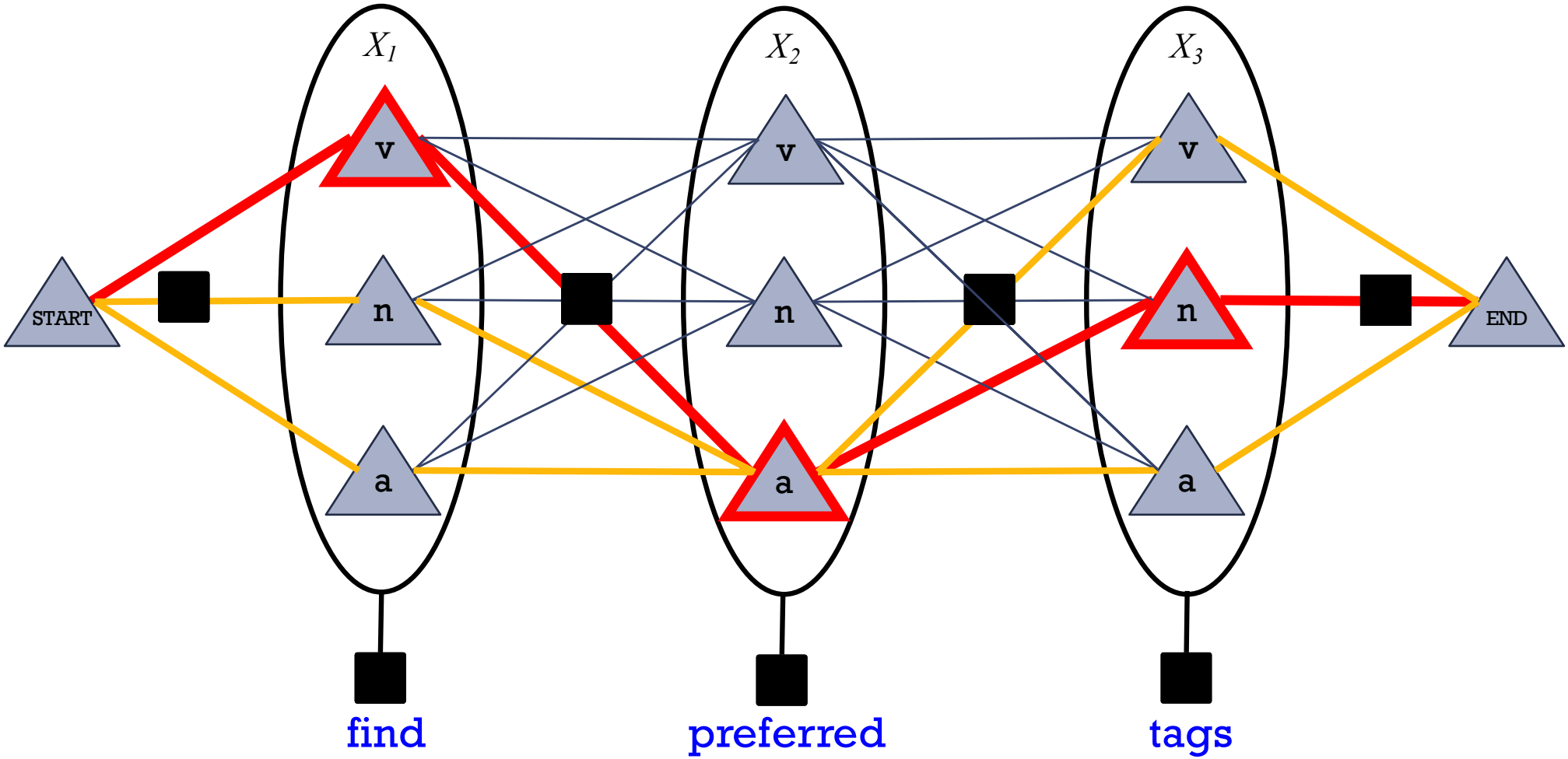
- So p(**v a n**) = (1/Z) * product of 7 numbers
- Numbers associated with edges and nodes of path
- Most probable assignment = **path with highest product**

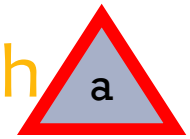
Viterbi Algorithm: Most Probable Assignment



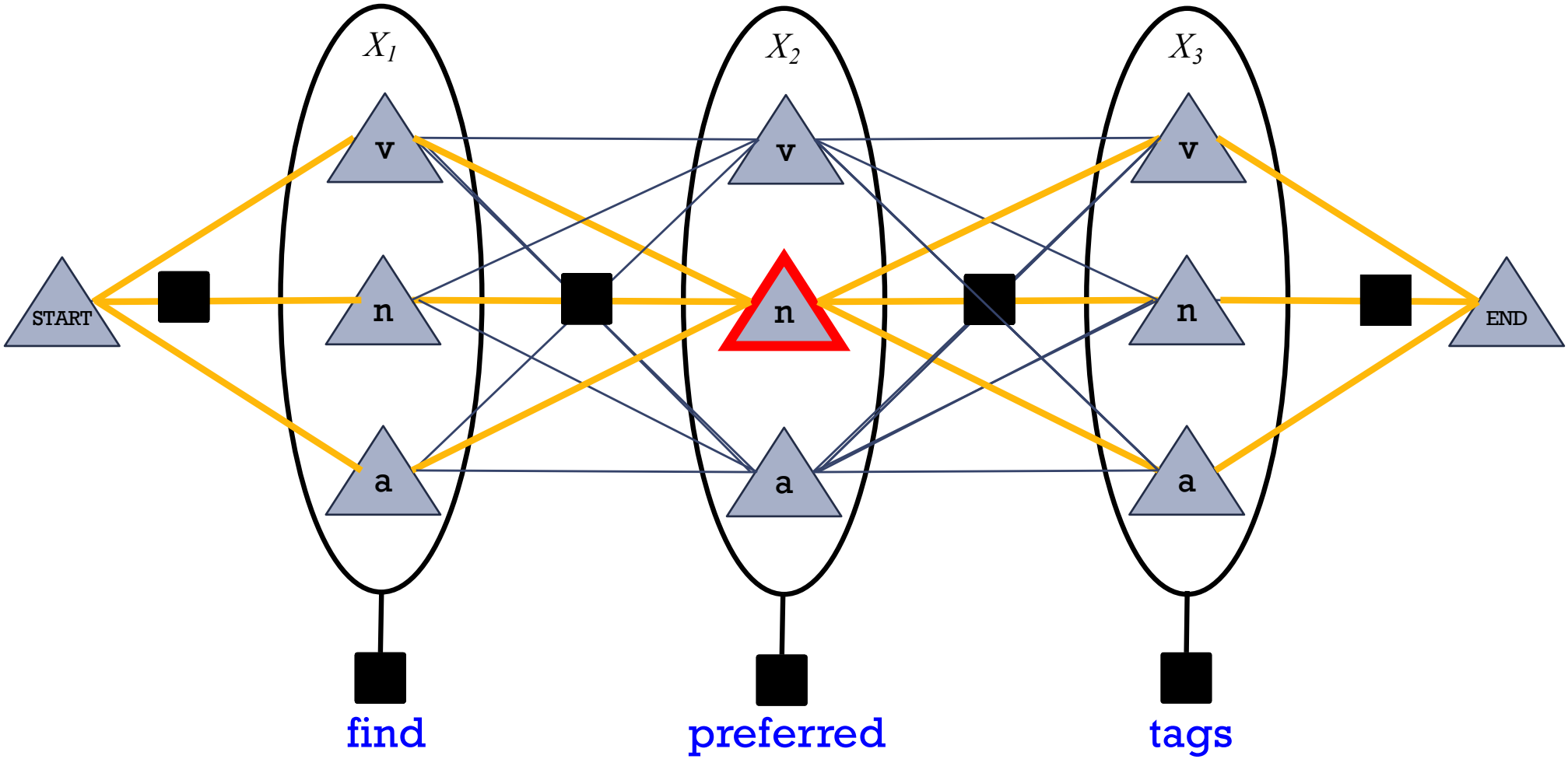
- So p(**v a n**) = (1/Z) * product weight of one path

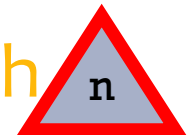
Forward-Backward Algorithm: Finds Marginals



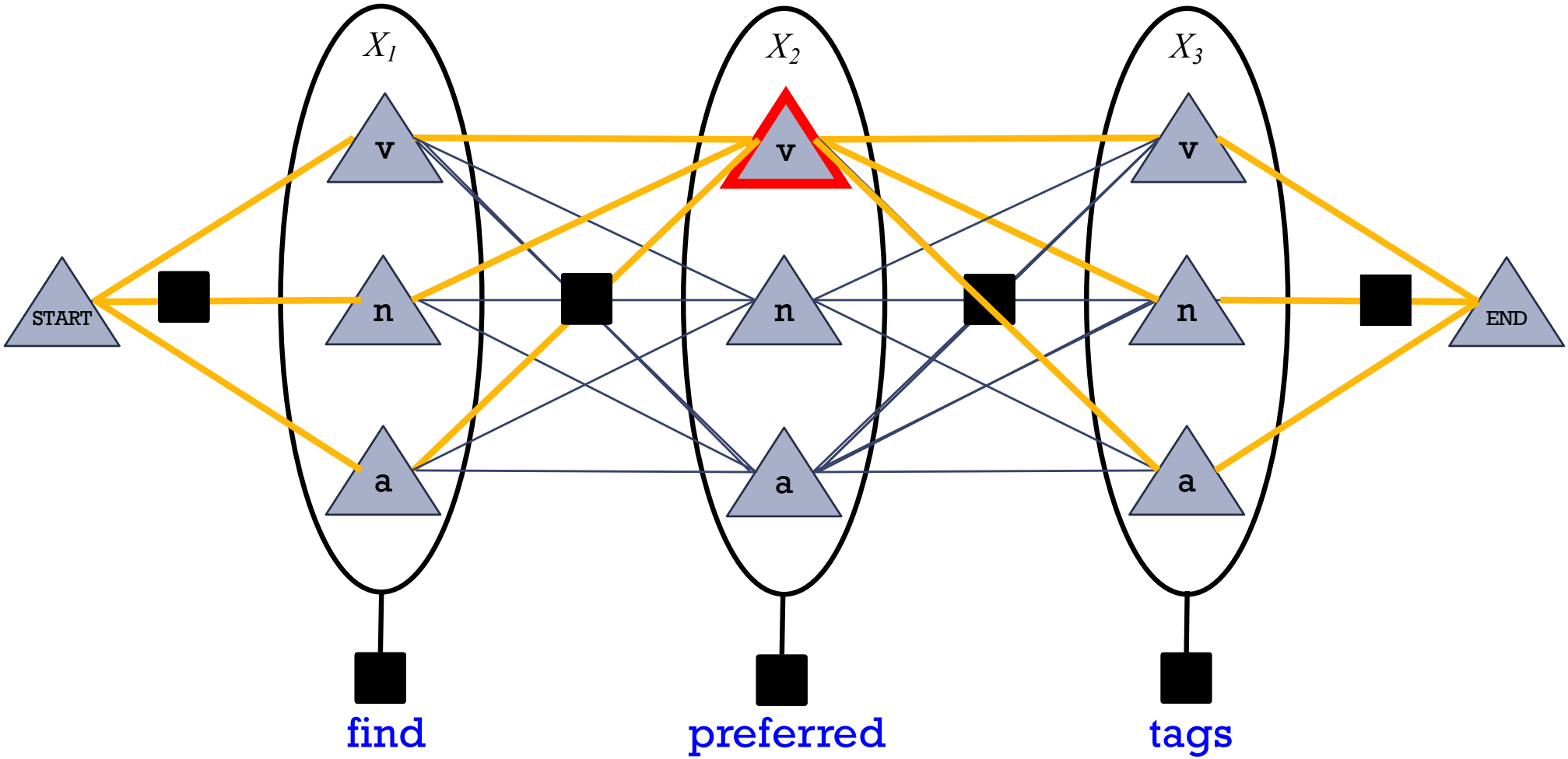
- So p(**v a n**) = (1/Z) * product weight of one path
- Marginal probability p(X_2 = a)
  = (1/Z) * total weight of *all* paths through 

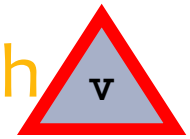
Forward-Backward Algorithm: Finds Marginals



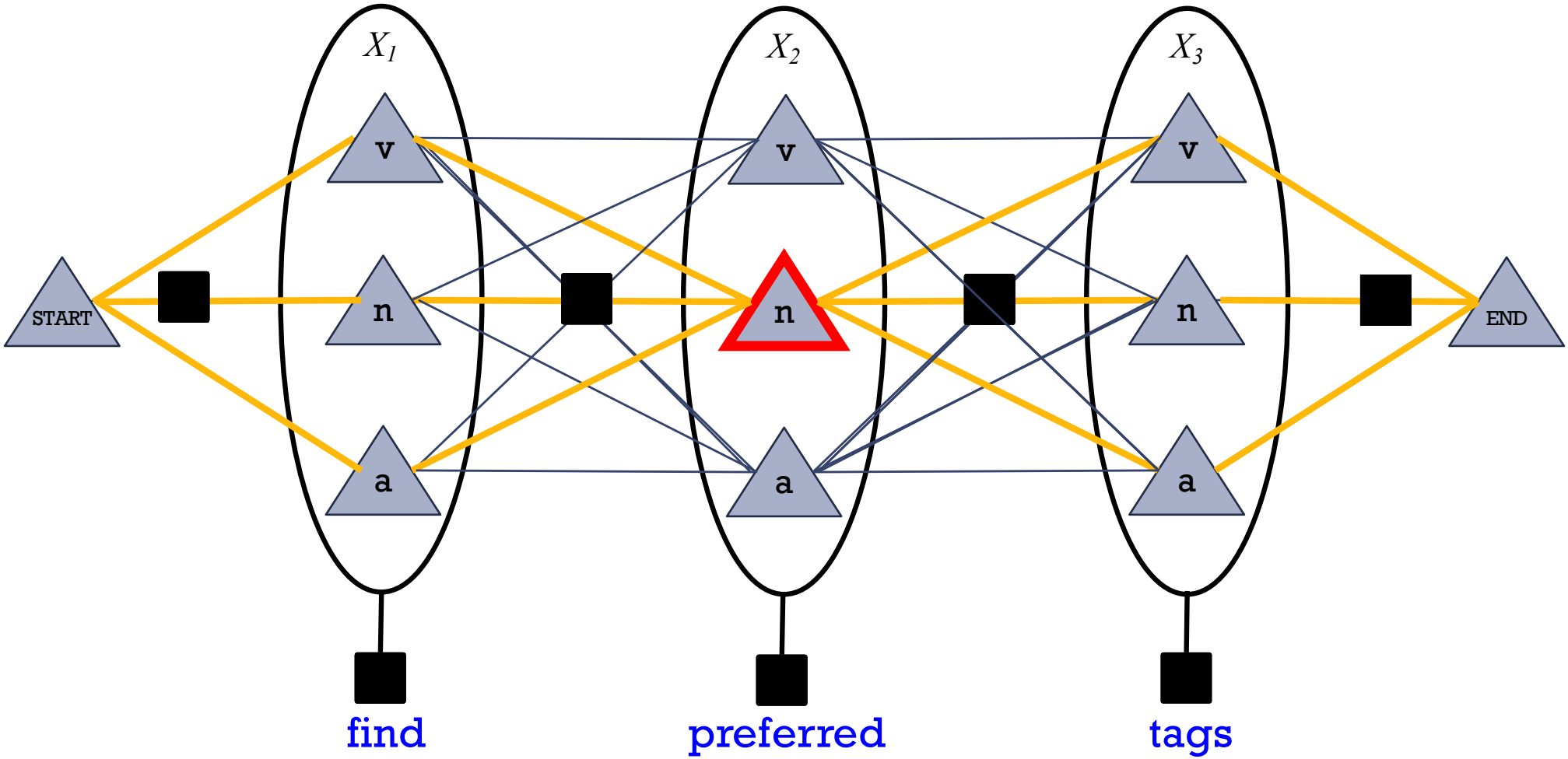
- So p(**v a n**) = (1/Z) * product weight of one path
- Marginal probability p(X_2 = a)
  = (1/Z) * total weight of *all* paths through 

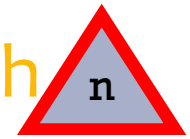
Forward-Backward Algorithm: Finds Marginals



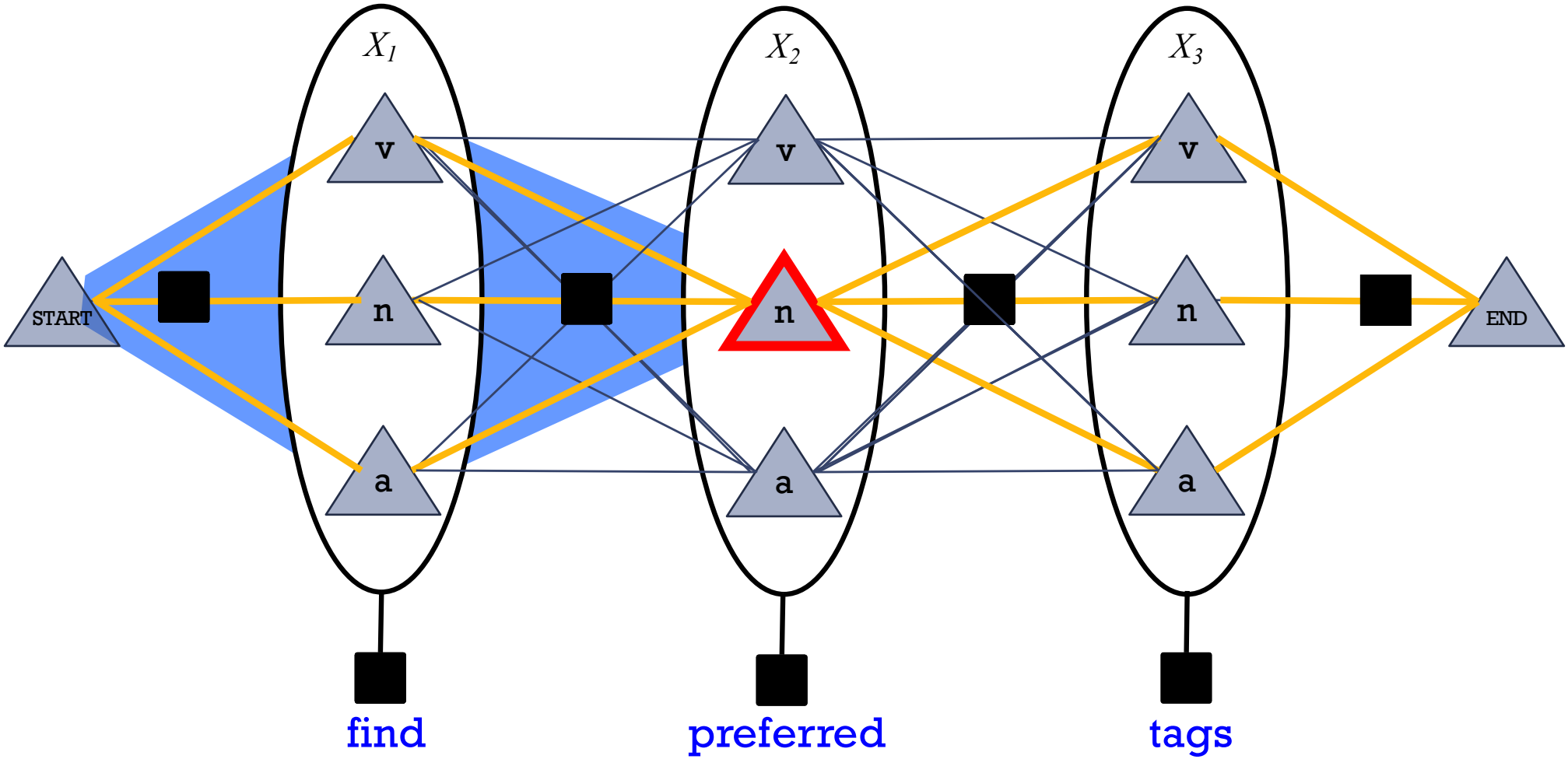
- So p(**v a n**) = (1/Z) * product weight of one path
- Marginal probability p(X_2 = a)
  = (1/Z) * total weight of *all* paths through 

Forward-Backward Algorithm: Finds Marginals



- So p(**v a n**) = (1/Z) * product weight of one path
- Marginal probability p(X_2 = a)
  = (1/Z) * total weight of *all* paths through 

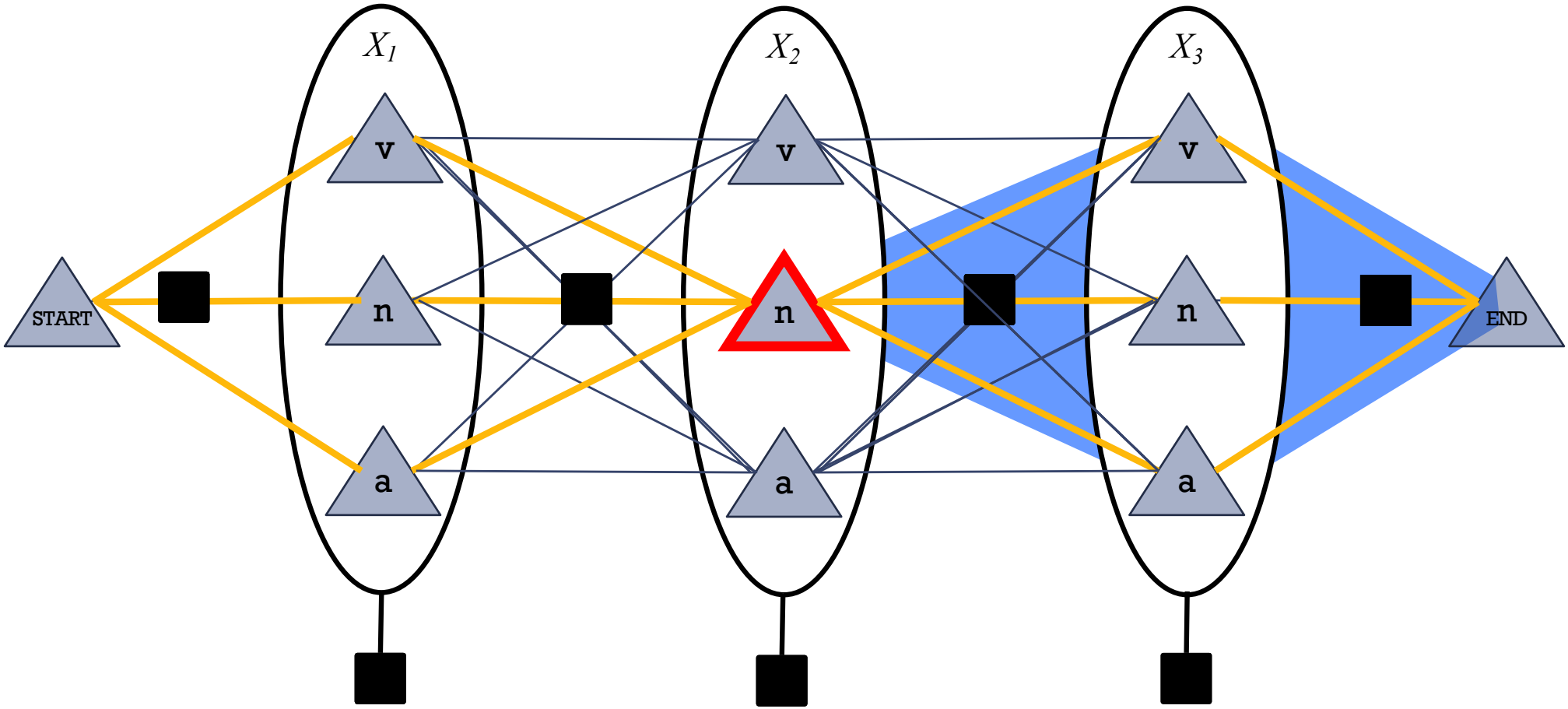
Forward-Backward Algorithm: Finds Marginals



$\alpha_2(\mathbf{n})$ = total weight of these path *prefixes*

(found by dynamic programming: matrix-vector products)

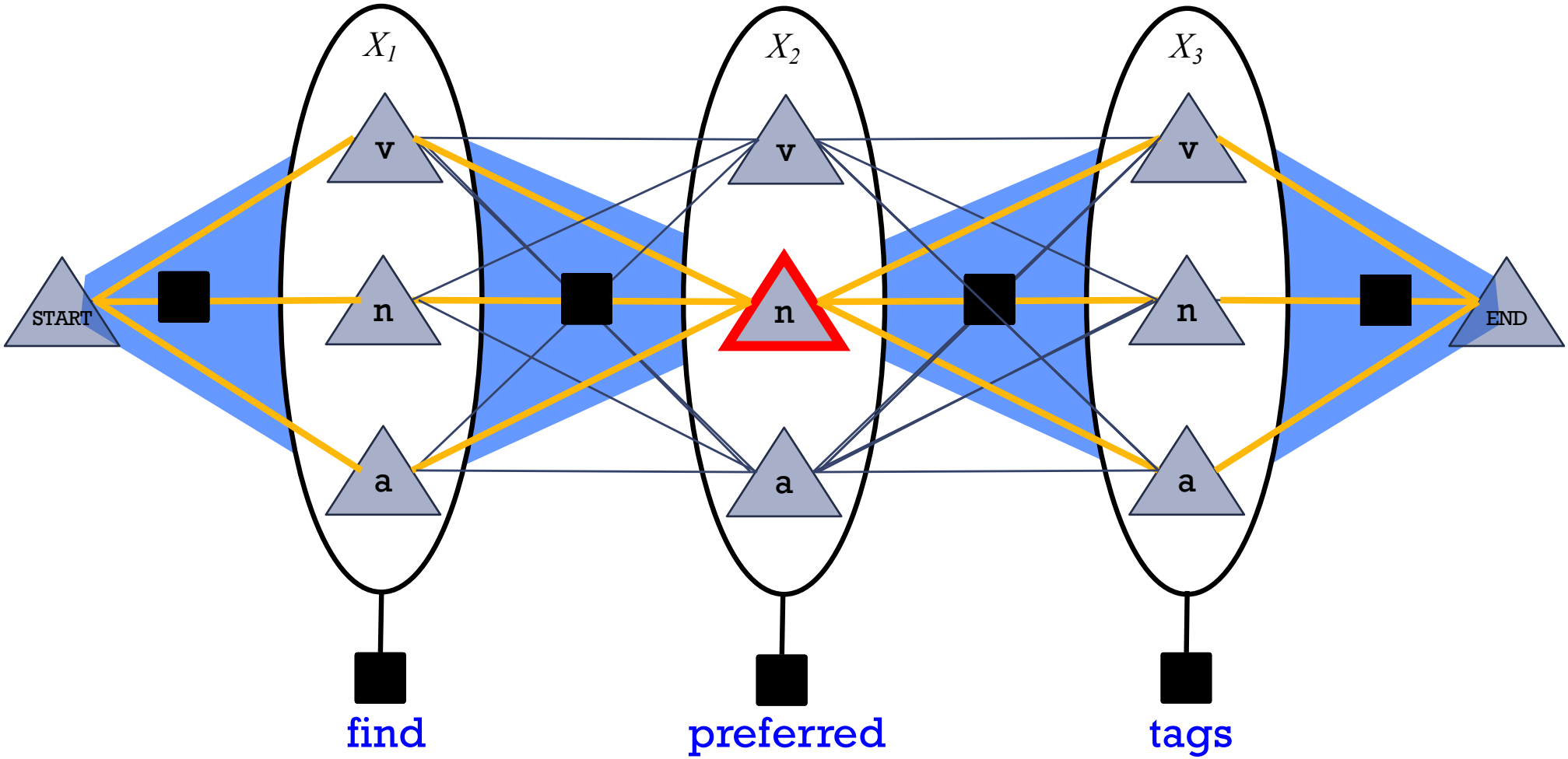
Forward-Backward Algorithm: Finds Marginals



$\beta_2(\mathbf{n})$ = total weight of these path *suffixes*

(found by dynamic programming: matrix-vector products)

Forward-Backward Algorithm: Finds Marginals



$\alpha_2(\mathbf{n})$ = total weight of these path *prefixes* (a + b + c)

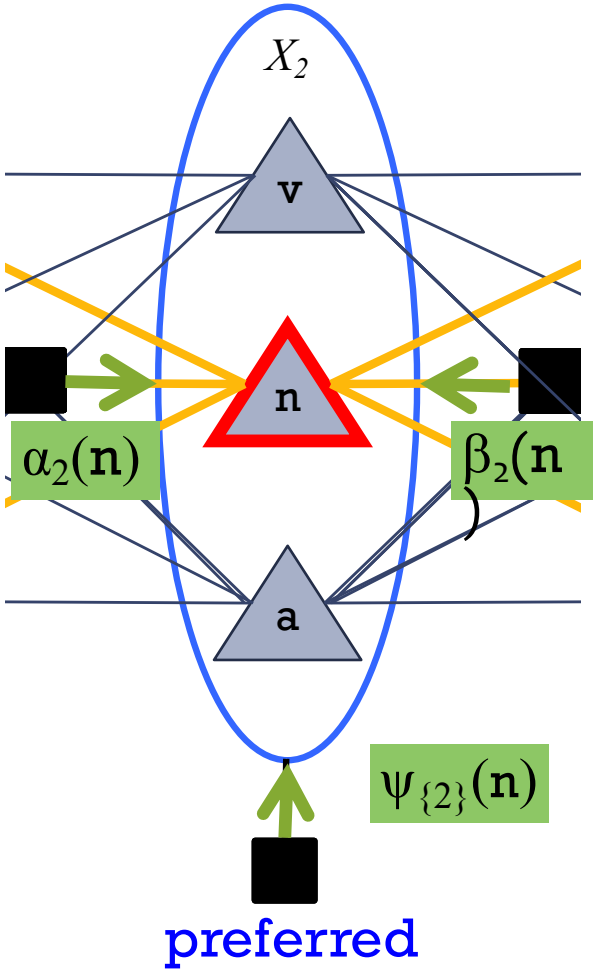
$\beta_2(\mathbf{n})$ = total weight of these path *suffixes* (x + y + z)

Product gives ax+ay+az+bx+by+bz+cx+cy+cz = total weight of paths

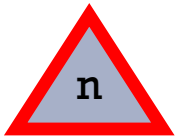
Forward-Backward Algorithm: Finds Marginals

Oops! The weight of a path through a state also includes a weight at that state.
So $\alpha(\mathbf{n})\cdot\beta(\mathbf{n})$ isn't enough.

The extra weight is the opinion of the unigram factor at this variable.

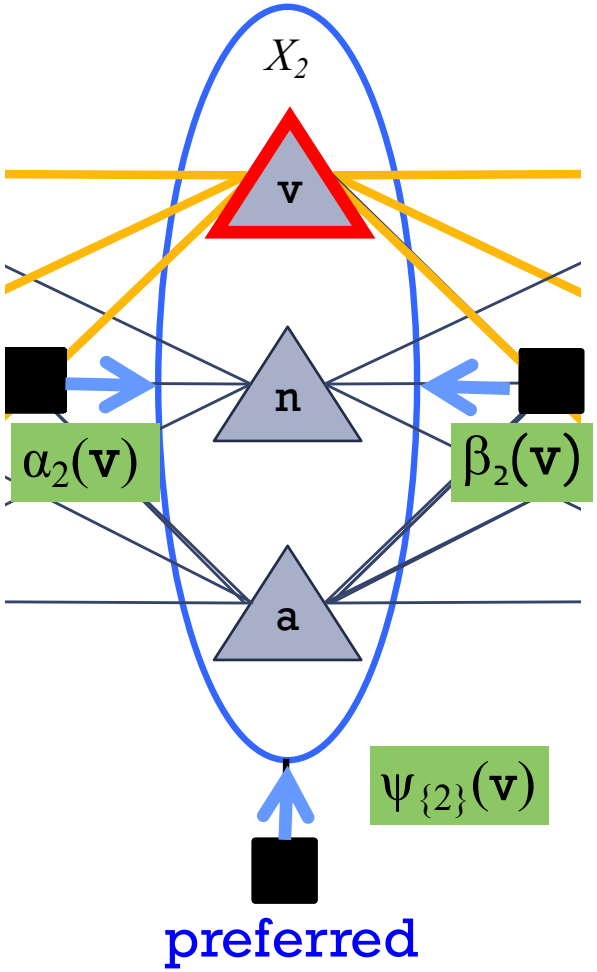


"belief that $X_2 = \mathbf{n}$"

total weight of *all* paths through 

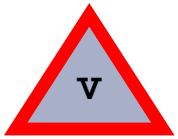
= $\alpha_2(\mathbf{n})$ $\psi_{\{2\}}(\mathbf{n})$ $\beta_2(\mathbf{n})$

Forward-Backward Algorithm: Finds Marginals



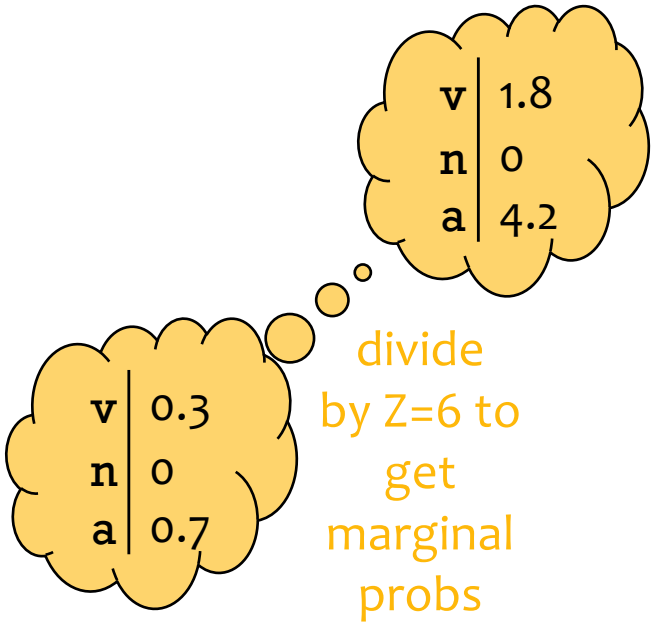
"belief that X_2 = **v**"

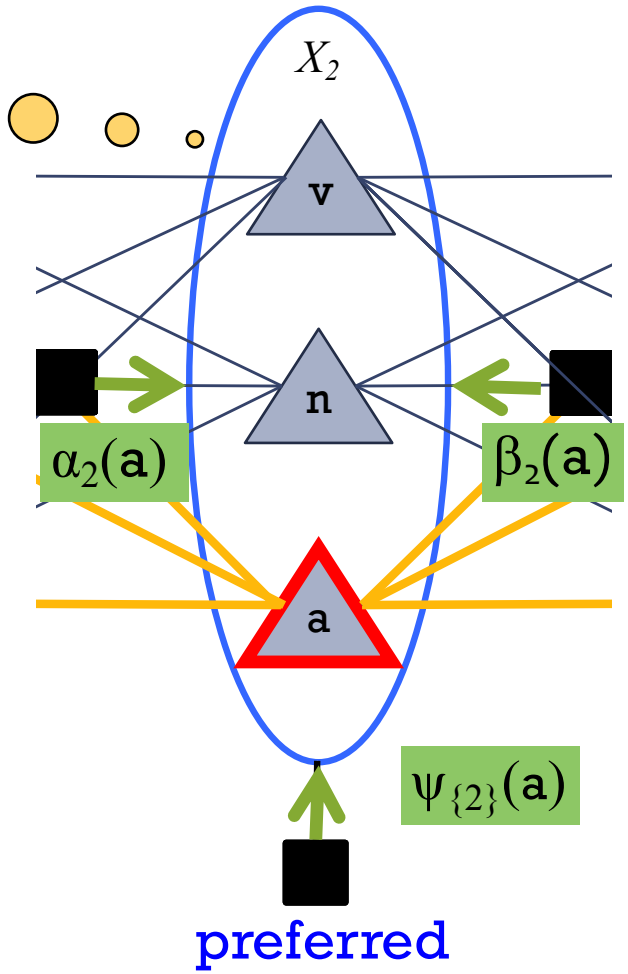
"belief that X_2 = **n**"

total weight of *all* paths through 

= $\alpha_2(\mathbf{v})$ $\psi_{\{2\}}(\mathbf{v})$ $\beta_2(\mathbf{v})$

Forward-Backward Algorithm: Finds Marginals





"belief that X_2 = **v**"

"belief that X_2 = **n**"

"belief that X_2 = **a**"

---

sum = Z
(total probability of *all* paths)

total weight of *all* paths through **a**

= $\alpha_2(a)$ $\psi_{\{2\}}(a)$ $\beta_2(a)$

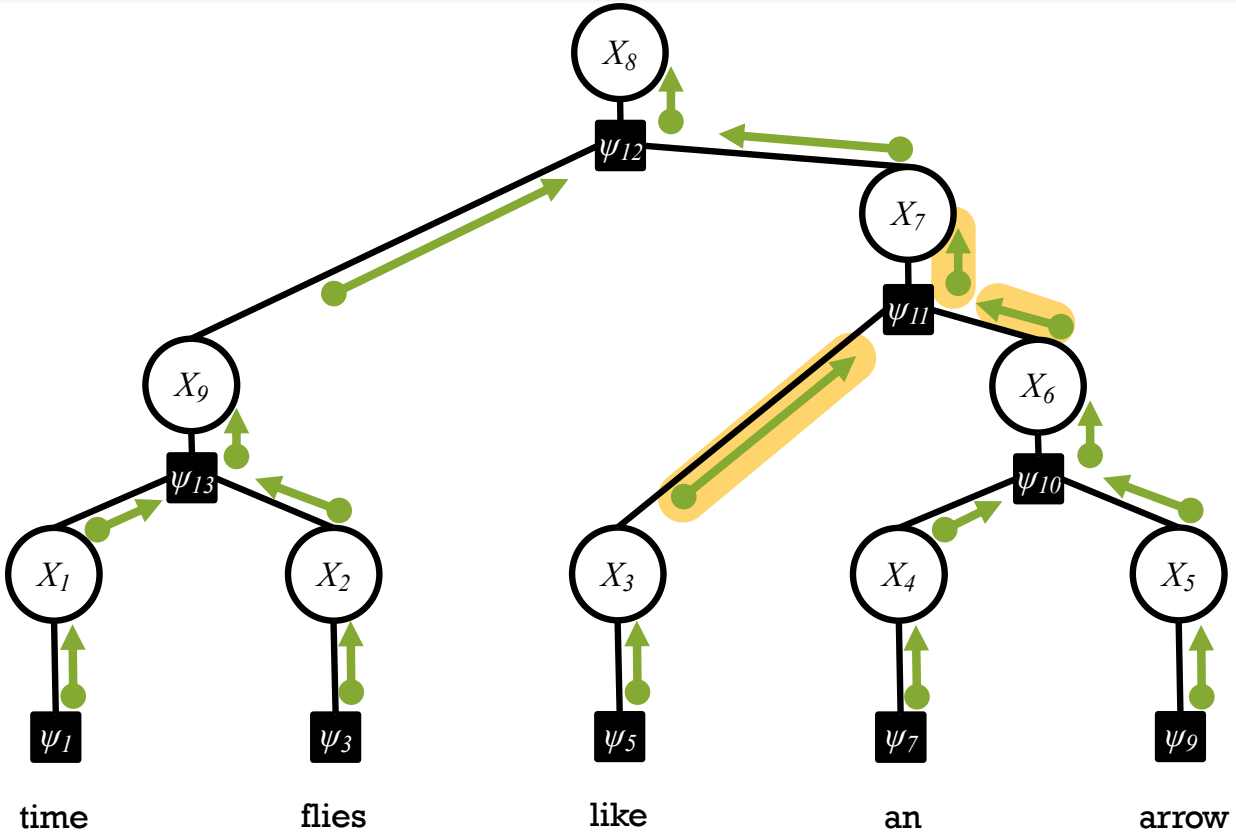
BP AS DYNAMIC PROGRAMMING

(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.

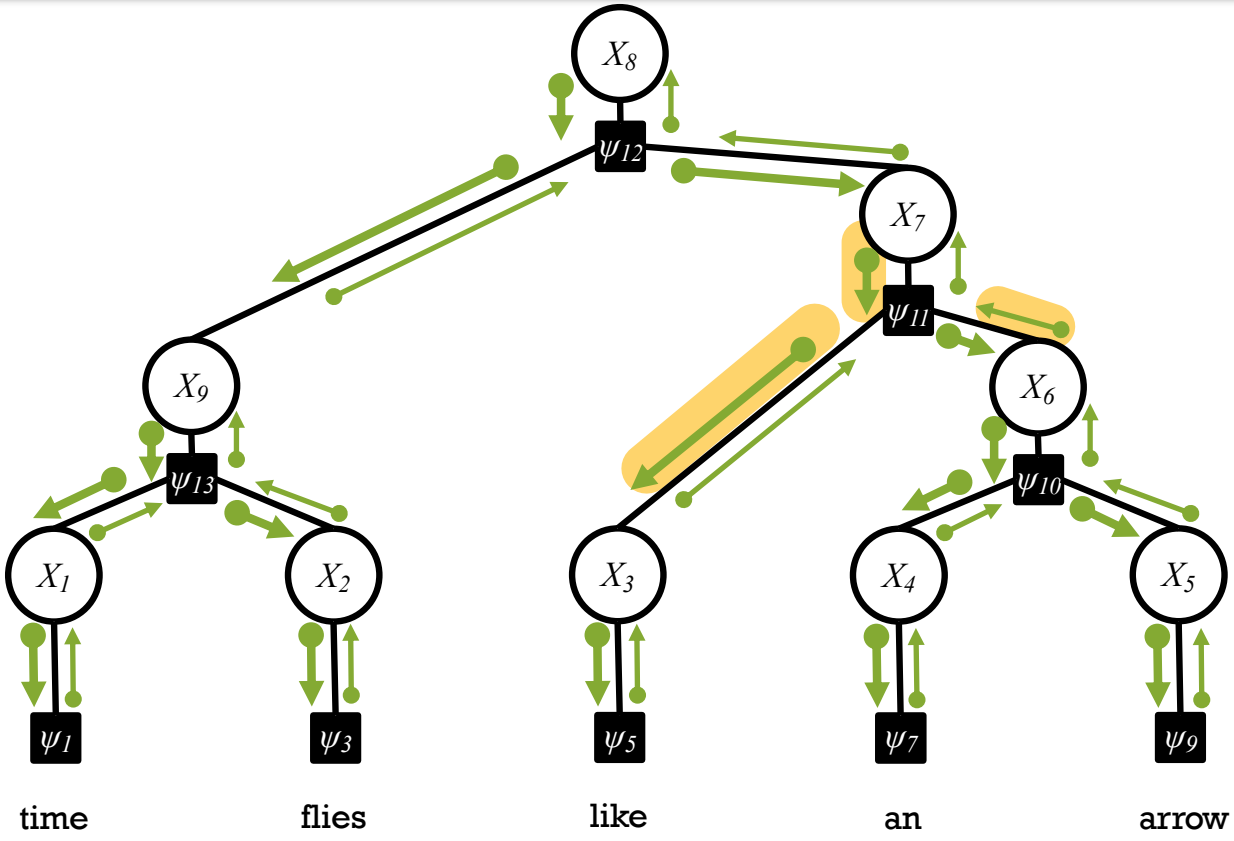


(Acyclic) Belief Propagation

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**.

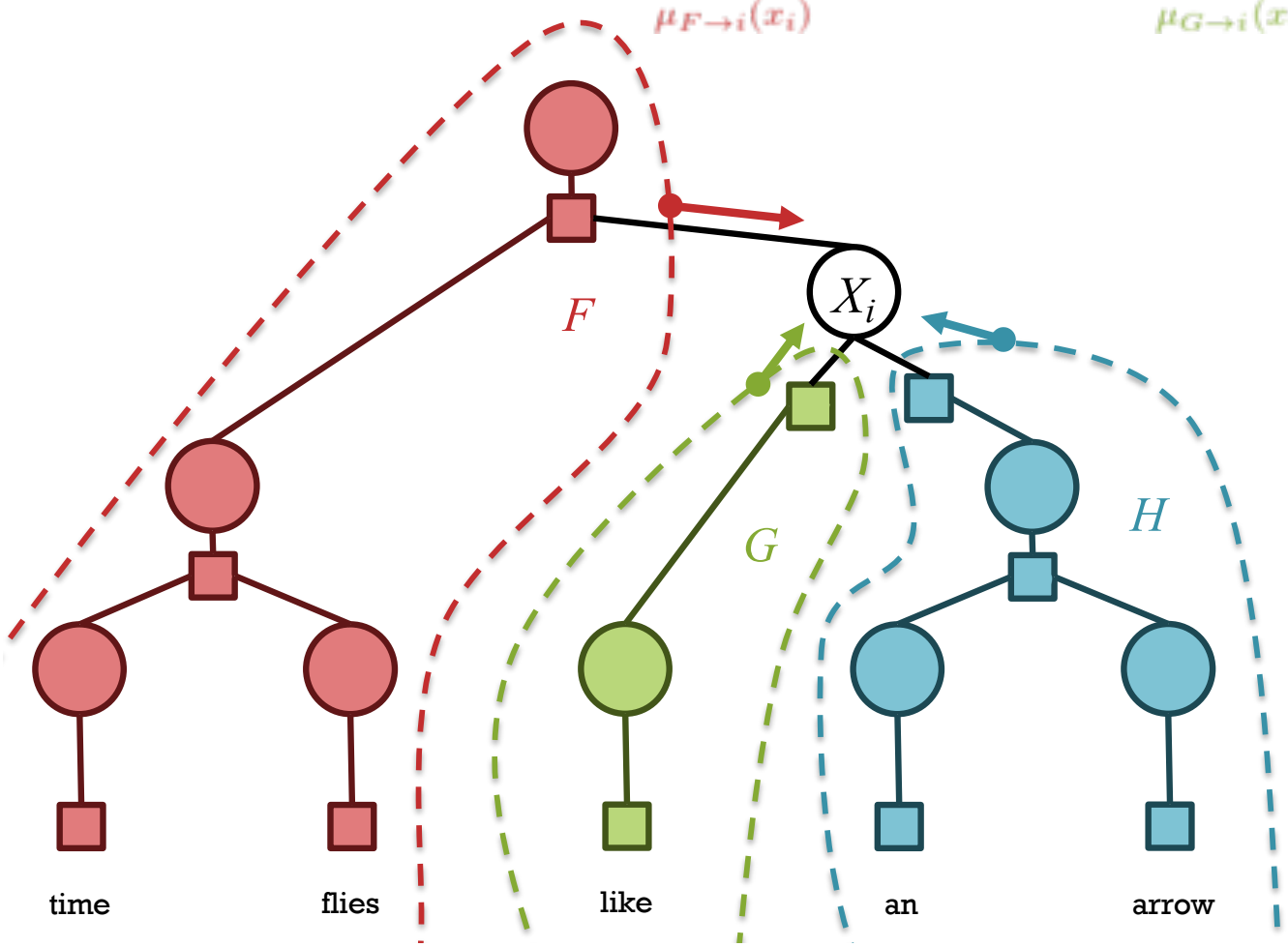
A node computes an outgoing message along an edge only after it has received incoming messages along all its other edges.



Acyclic BP as Dynamic Programming

$$p(X_i = x_i) \propto b_i(x_i) = \sum_{\boldsymbol{x}:\boldsymbol{x}[i]=x_i} \prod_{\alpha} \psi_\alpha(\boldsymbol{x}_\alpha)$$

$$= \underbrace{\left(\sum_{\boldsymbol{x}:\boldsymbol{x}[i]=x_i} \prod_{\alpha \subseteq F} \psi_\alpha(\boldsymbol{x}_\alpha) \right)}_{\mu_{F \to i}(x_i)} \underbrace{\left(\sum_{\boldsymbol{x}:\boldsymbol{x}[i]=x_i} \prod_{\alpha \subseteq G} \psi_\alpha(\boldsymbol{x}_\alpha) \right)}_{\mu_{G \to i}(x_i)} \underbrace{\left(\sum_{\boldsymbol{x}:\boldsymbol{x}[i]=x_i} \prod_{\alpha \subseteq H} \psi_\alpha(\boldsymbol{x}_\alpha) \right)}_{\mu_{H \to i}(x_i)}$$



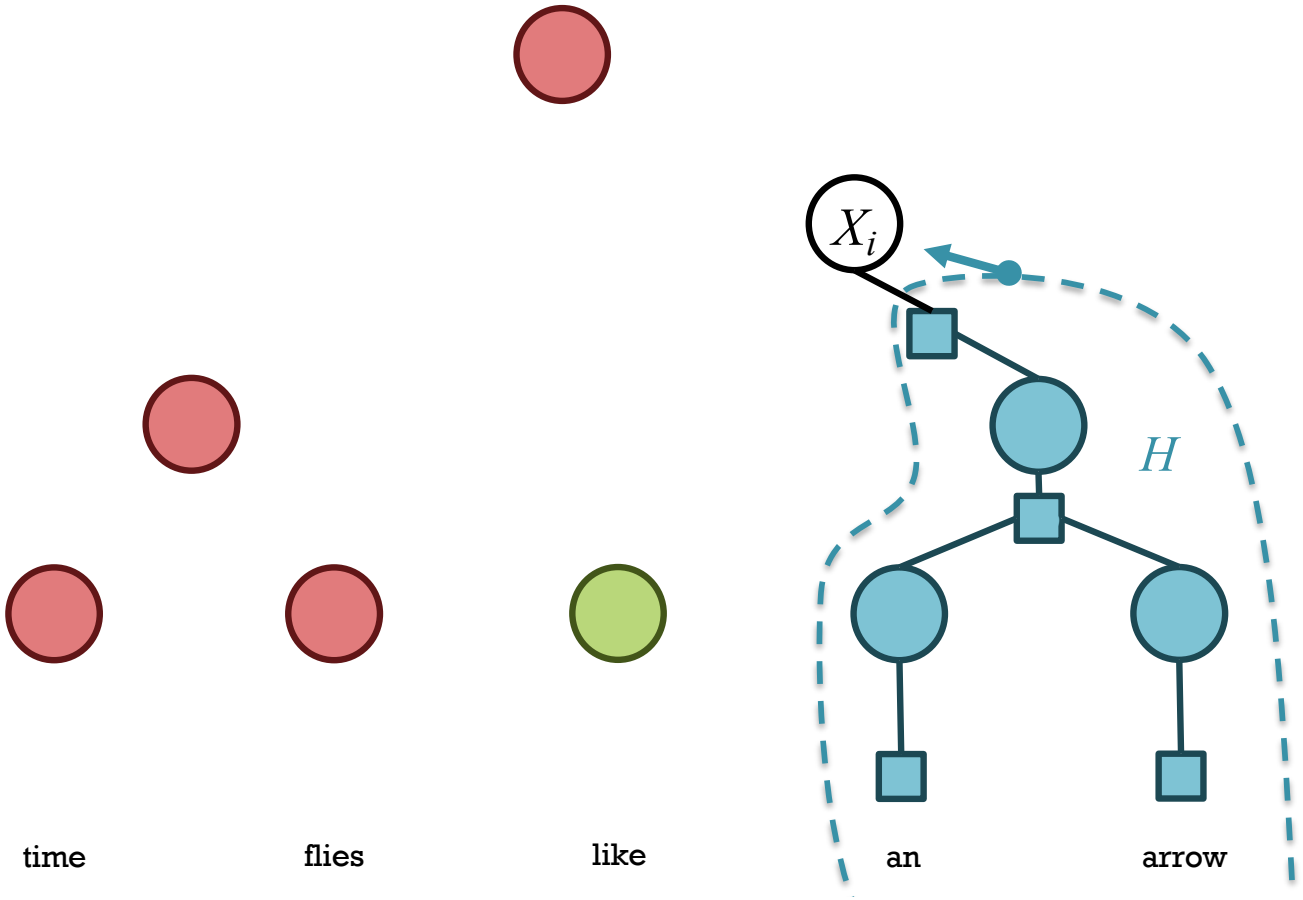
Subproblem:
Inference using just the factors in subgraph H

Figure adapted from Burkett & Klein (2012)

Acyclic BP as Dynamic Programming

$$p(X_i = x_i) \propto b_i(x_i) = \sum_{\boldsymbol{x}:\boldsymbol{x}[i]=x_i} \prod_{\alpha} \psi_\alpha(\boldsymbol{x}_\alpha)$$

$$= \underbrace{\left(\sum_{\boldsymbol{x}:\boldsymbol{x}[i]=x_i} \prod_{\alpha \subseteq F} \psi_\alpha(\boldsymbol{x}_\alpha) \right)}_{\mu_{F \to i}(x_i)} \underbrace{\left(\sum_{\boldsymbol{x}:\boldsymbol{x}[i]=x_i} \prod_{\alpha \subseteq G} \psi_\alpha(\boldsymbol{x}_\alpha) \right)}_{\mu_{G \to i}(x_i)} \underbrace{\left(\sum_{\boldsymbol{x}:\boldsymbol{x}[i]=x_i} \prod_{\alpha \subseteq H} \psi_\alpha(\boldsymbol{x}_\alpha) \right)}_{\mu_{H \to i}(x_i)}$$



Subproblem:
Inference using just the factors in subgraph H

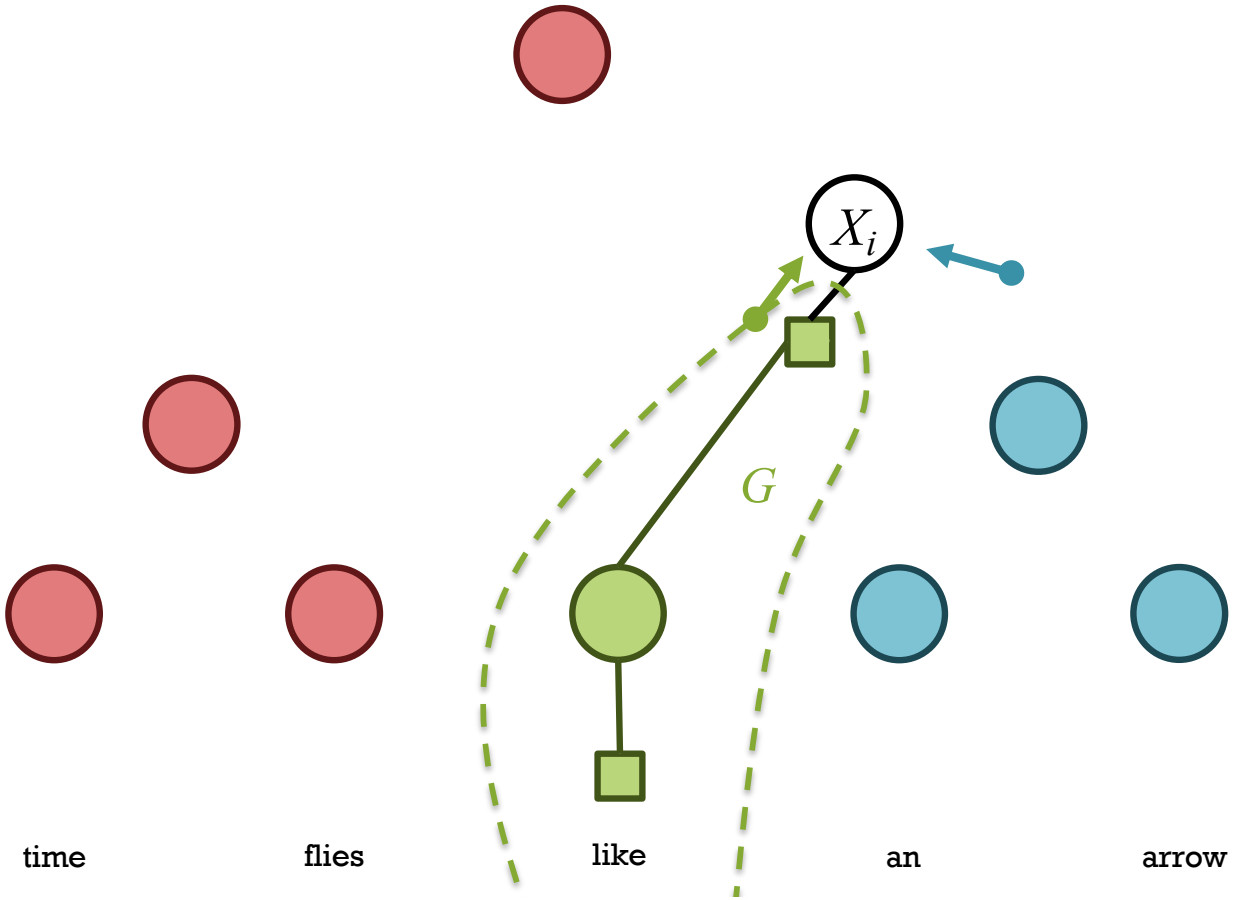
The marginal of X_i in that smaller model is the message sent to X_i from subgraph H

*Message **to** a variable*

Acyclic BP as Dynamic Programming

$$p(X_i = x_i) \propto b_i(x_i) = \sum_{\boldsymbol{x}:\boldsymbol{x}[i]=x_i} \prod_{\alpha} \psi_\alpha(\boldsymbol{x}_\alpha)$$

$$= \underbrace{\left(\sum_{\boldsymbol{x}:\boldsymbol{x}[i]=x_i} \prod_{\alpha \subseteq F} \psi_\alpha(\boldsymbol{x}_\alpha) \right)}_{\mu_{F \to i}(x_i)} \underbrace{\left(\sum_{\boldsymbol{x}:\boldsymbol{x}[i]=x_i} \prod_{\alpha \subseteq G} \psi_\alpha(\boldsymbol{x}_\alpha) \right)}_{\mu_{G \to i}(x_i)} \underbrace{\left(\sum_{\boldsymbol{x}:\boldsymbol{x}[i]=x_i} \prod_{\alpha \subseteq H} \psi_\alpha(\boldsymbol{x}_\alpha) \right)}_{\mu_{H \to i}(x_i)}$$



Subproblem:
Inference using just the factors in subgraph H

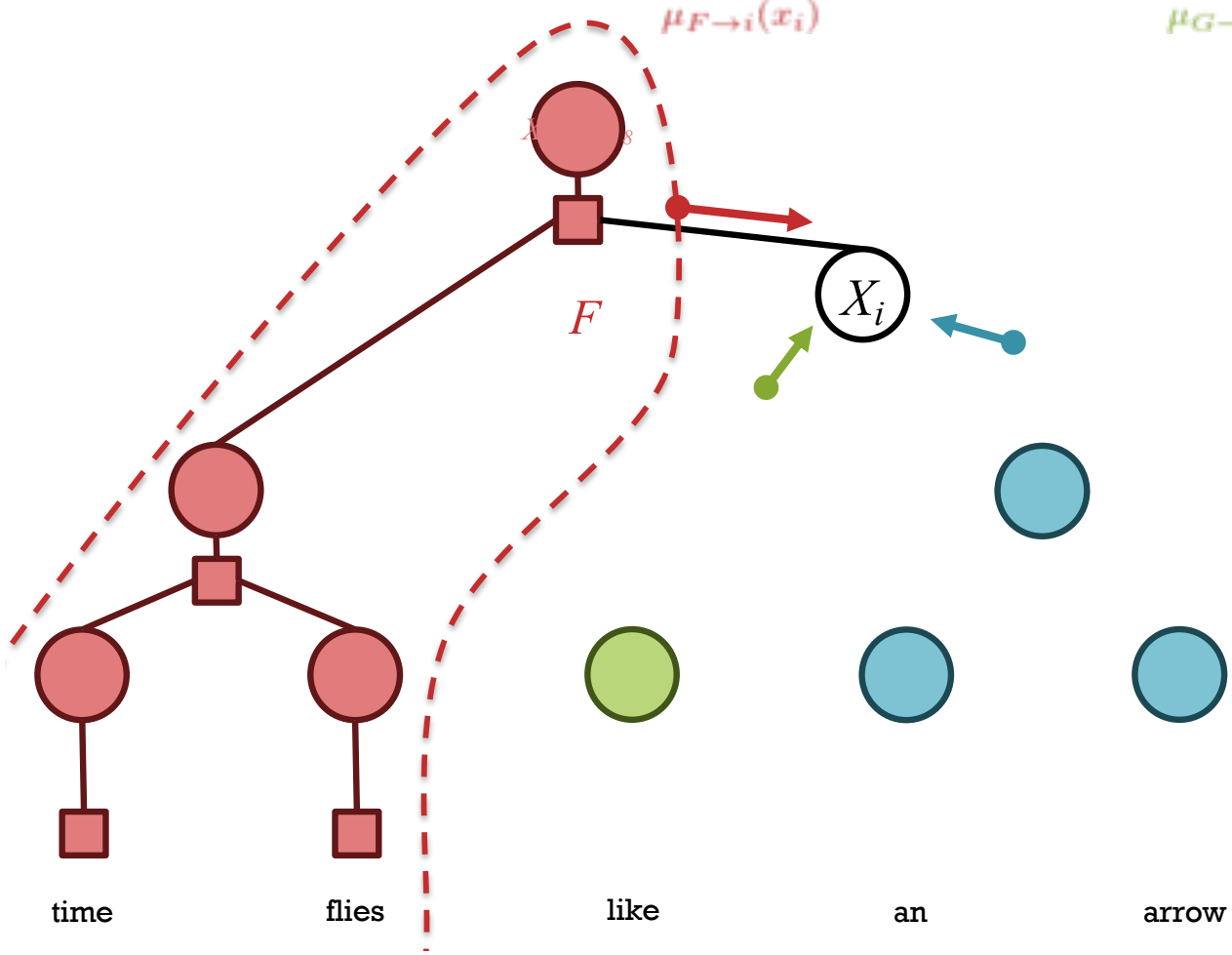
The marginal of X_i in that smaller model is the message sent to X_i from subgraph H

*Message **to** a variable*

Acyclic BP as Dynamic Programming

$$p(X_i = x_i) \propto b_i(x_i) = \sum_{\boldsymbol{x}:\boldsymbol{x}[i]=x_i} \prod_{\alpha} \psi_\alpha(\boldsymbol{x}_\alpha)$$

$$= \underbrace{\left(\sum_{\boldsymbol{x}:\boldsymbol{x}[i]=x_i} \prod_{\alpha \subseteq F} \psi_\alpha(\boldsymbol{x}_\alpha) \right)}_{\mu_{F \to i}(x_i)} \underbrace{\left(\sum_{\boldsymbol{x}:\boldsymbol{x}[i]=x_i} \prod_{\alpha \subseteq G} \psi_\alpha(\boldsymbol{x}_\alpha) \right)}_{\mu_{G \to i}(x_i)} \underbrace{\left(\sum_{\boldsymbol{x}:\boldsymbol{x}[i]=x_i} \prod_{\alpha \subseteq H} \psi_\alpha(\boldsymbol{x}_\alpha) \right)}_{\mu_{H \to i}(x_i)}$$



Subproblem:
Inference using just the factors in subgraph H

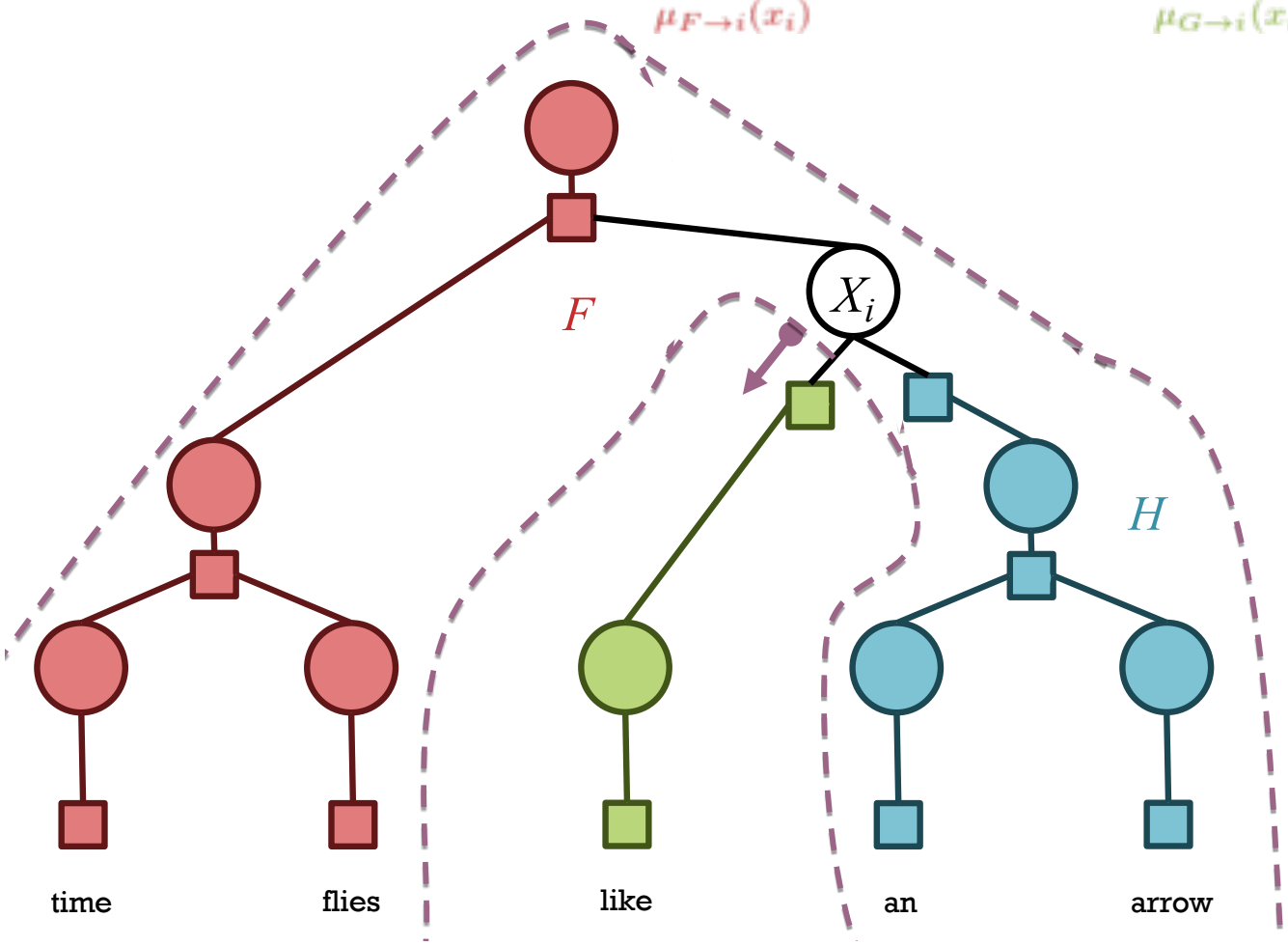
The marginal of X_i in that smaller model is the message sent to X_i from subgraph H

*Message **to** a variable*

Acyclic BP as Dynamic Programming

$$p(X_i = x_i) \propto b_i(x_i) = \sum_{\boldsymbol{x}:\boldsymbol{x}[i]=x_i} \prod_{\alpha} \psi_\alpha(\boldsymbol{x}_\alpha)$$

$$= \Big(\underbrace{\sum_{\boldsymbol{x}:\boldsymbol{x}[i]=x_i} \prod_{\alpha \subseteq F} \psi_\alpha(\boldsymbol{x}_\alpha)}_{\mu_{F \to i}(x_i)} \Big) \Big(\underbrace{\sum_{\boldsymbol{x}:\boldsymbol{x}[i]=x_i} \prod_{\alpha \subseteq G} \psi_\alpha(\boldsymbol{x}_\alpha)}_{\mu_{G \to i}(x_i)} \Big) \Big(\underbrace{\sum_{\boldsymbol{x}:\boldsymbol{x}[i]=x_i} \prod_{\alpha \subseteq H} \psi_\alpha(\boldsymbol{x}_\alpha)}_{\mu_{H \to i}(x_i)} \Big)$$



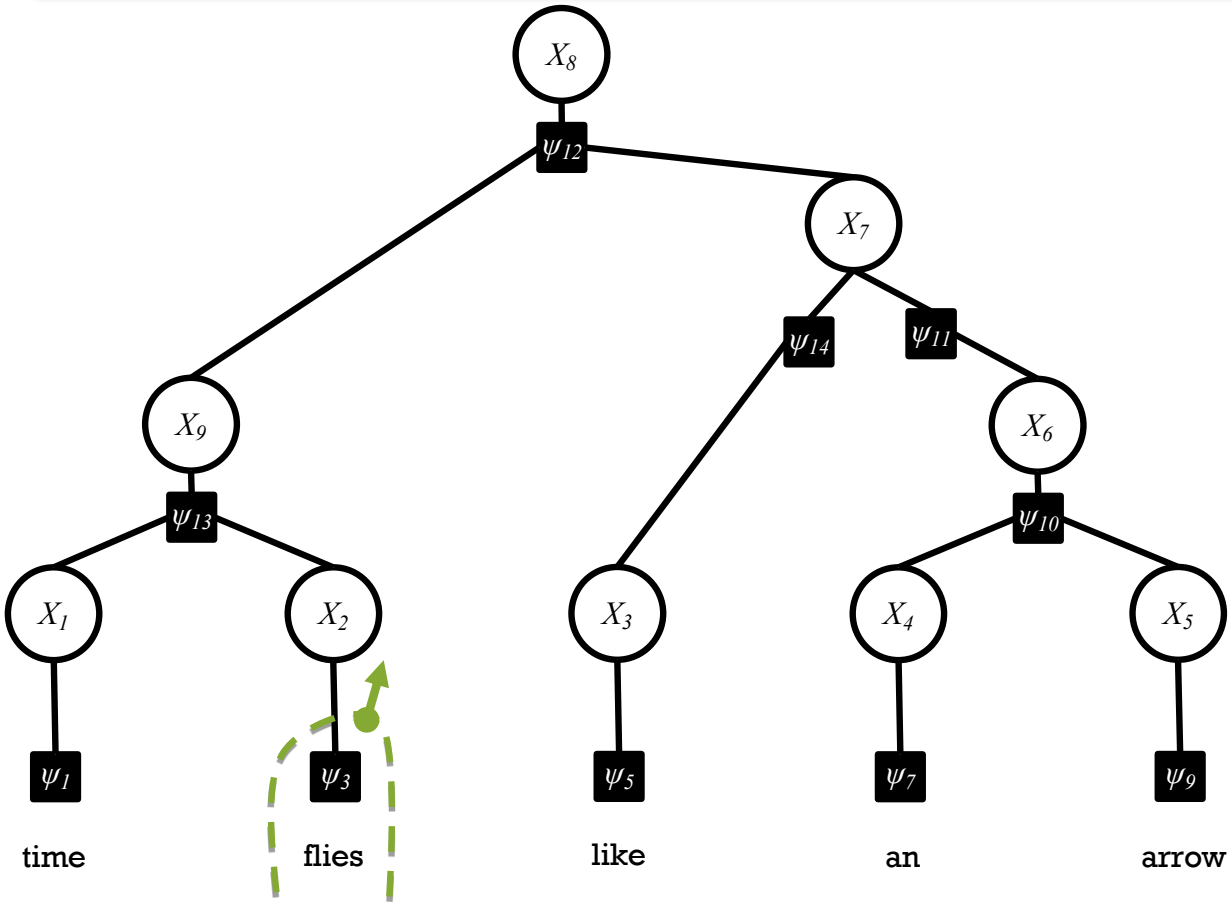
Subproblem:
Inference using just the factors in subgraph $F \cup H$

The marginal of X_i in that smaller model is the message sent by X_i out of subgraph $F \cup H$

*Message **from** a variable*

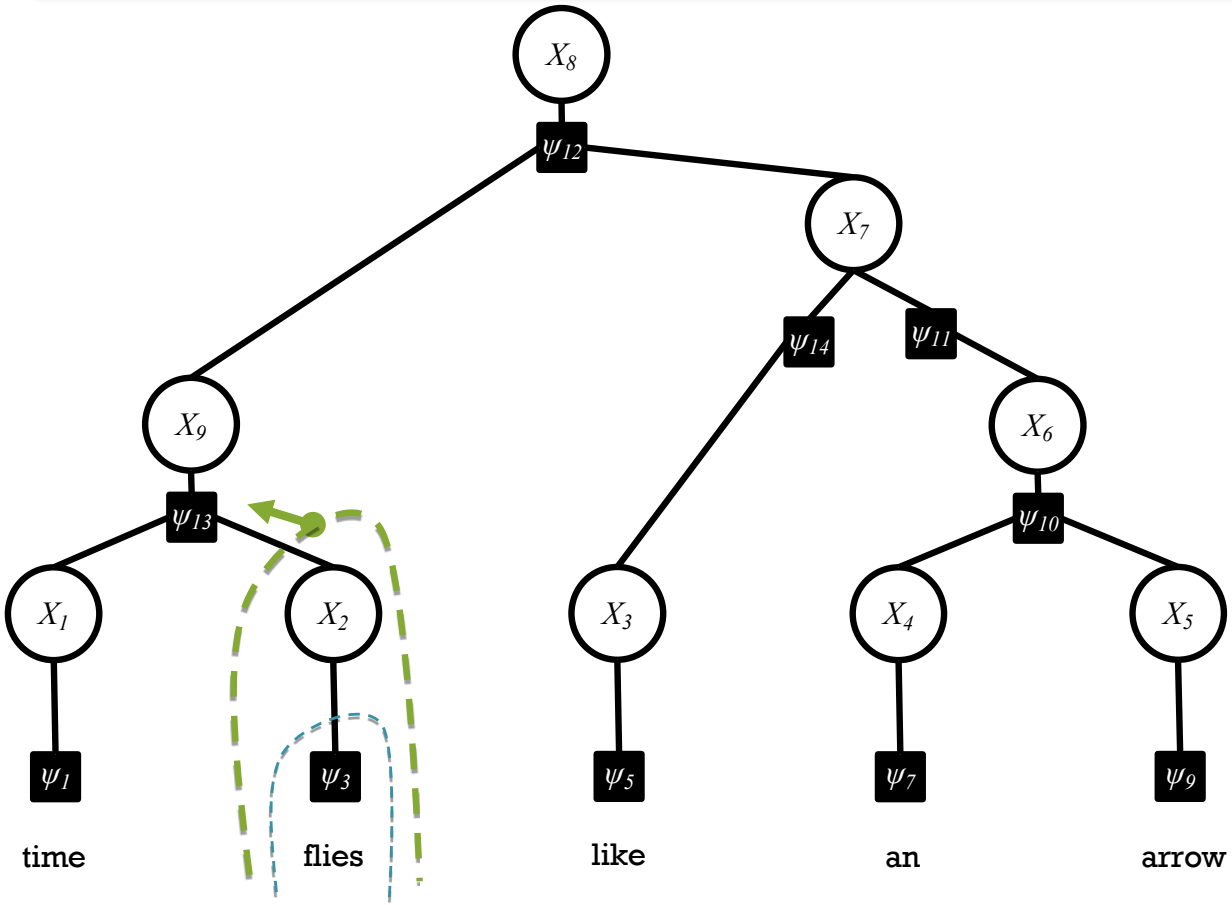
Acyclic BP as Dynamic Programming

- 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.



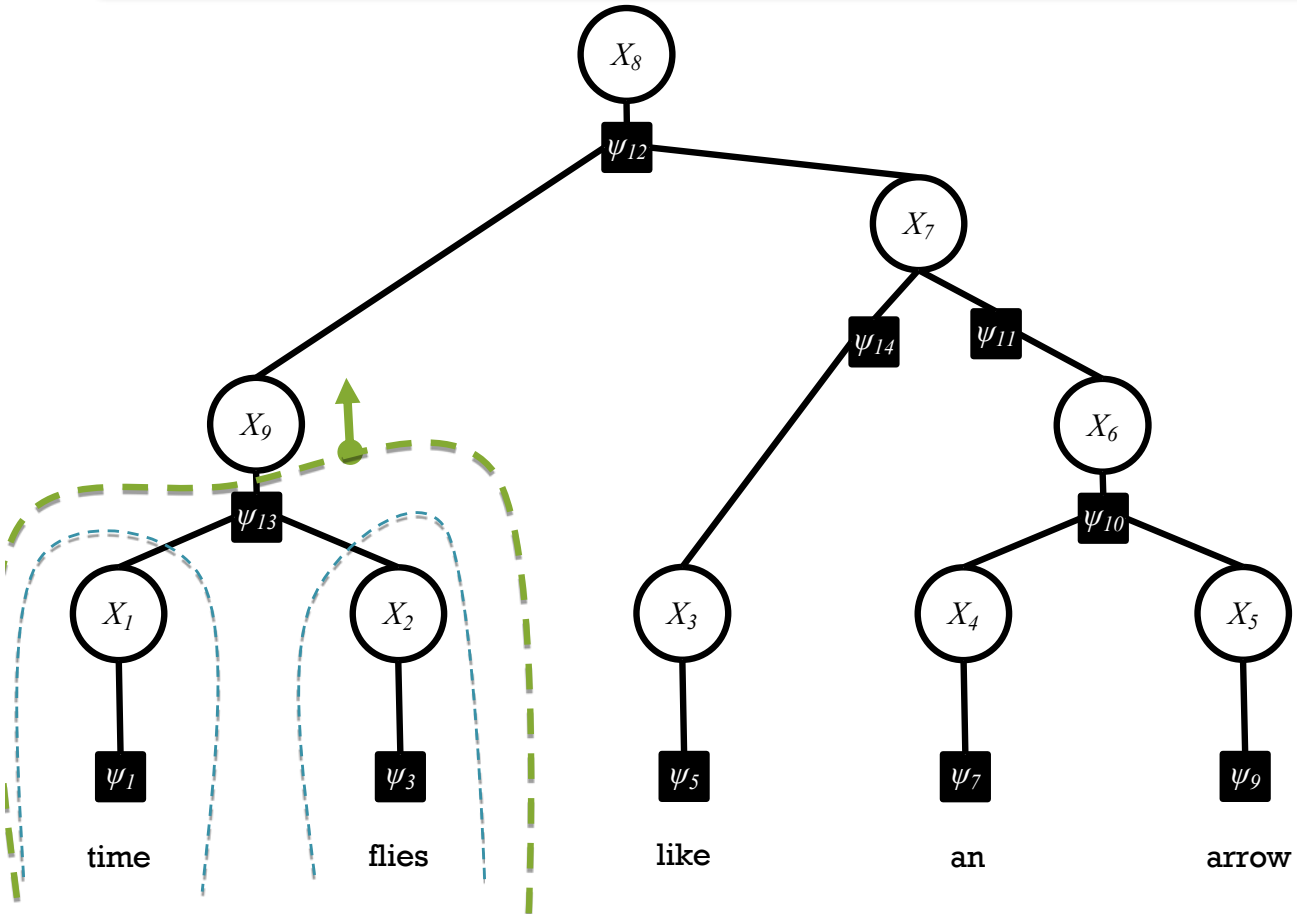
Acyclic BP as Dynamic Programming

- 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.



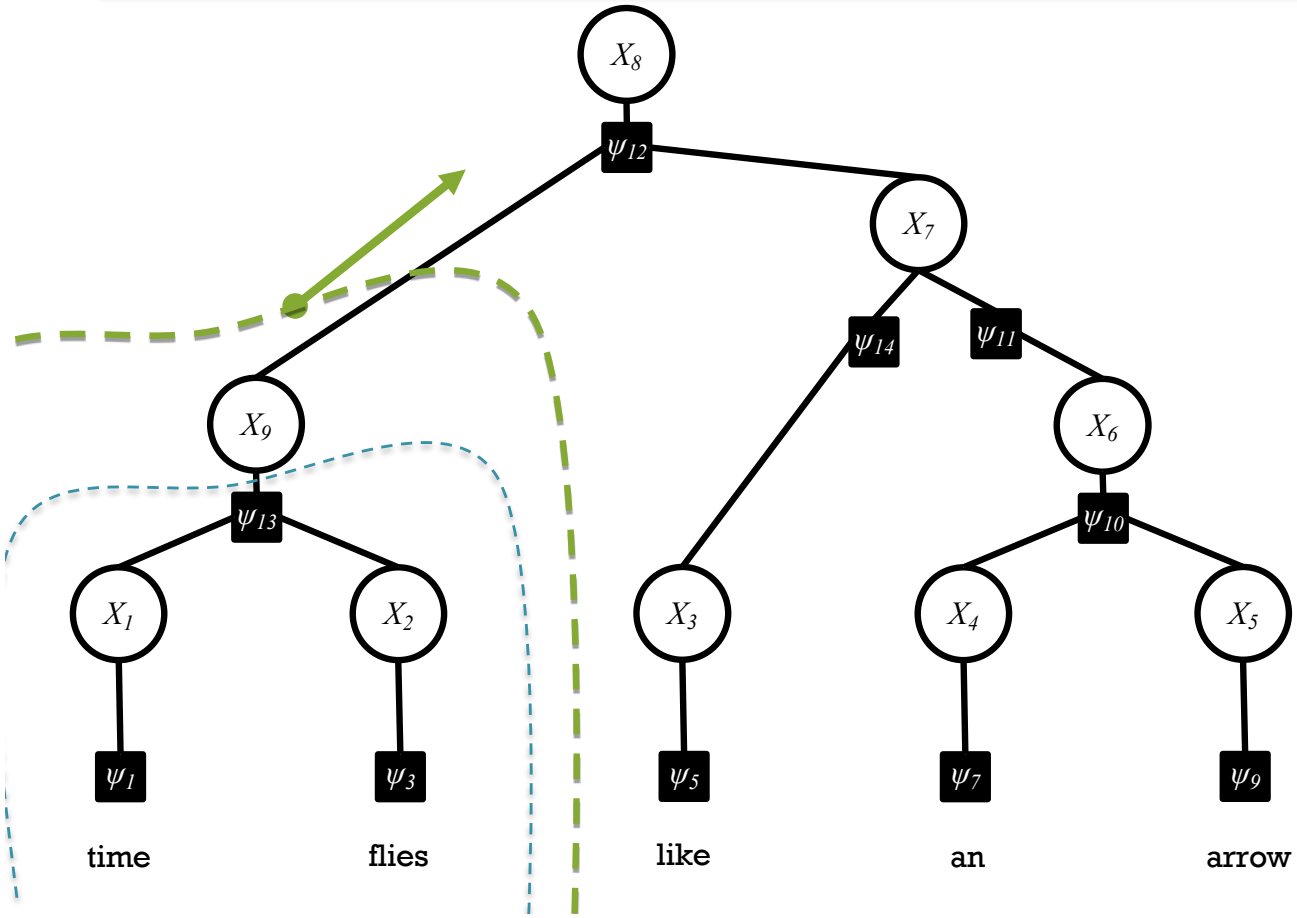
Acyclic BP as Dynamic Programming

- 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.



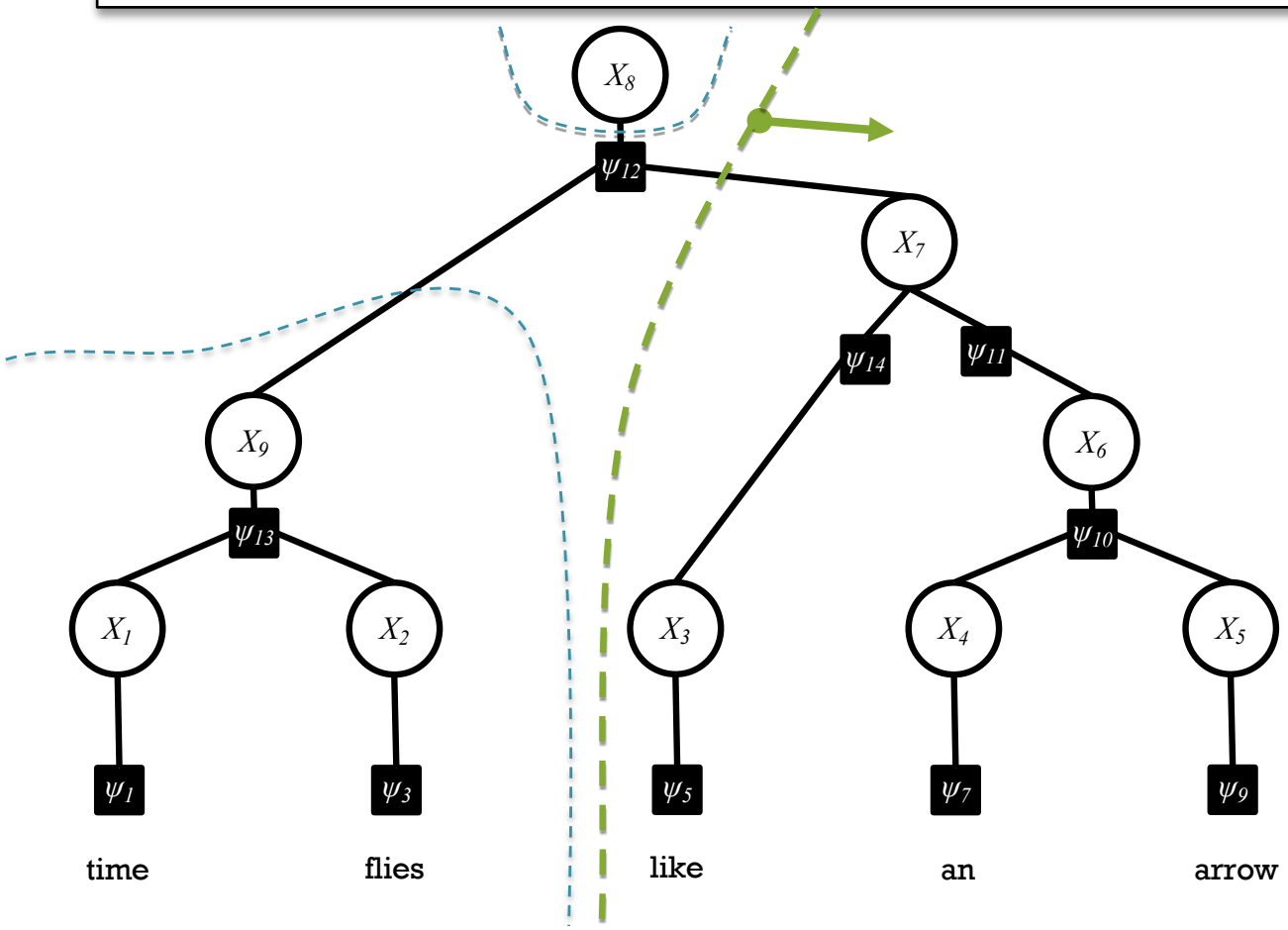
Acyclic BP as Dynamic Programming

- 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.



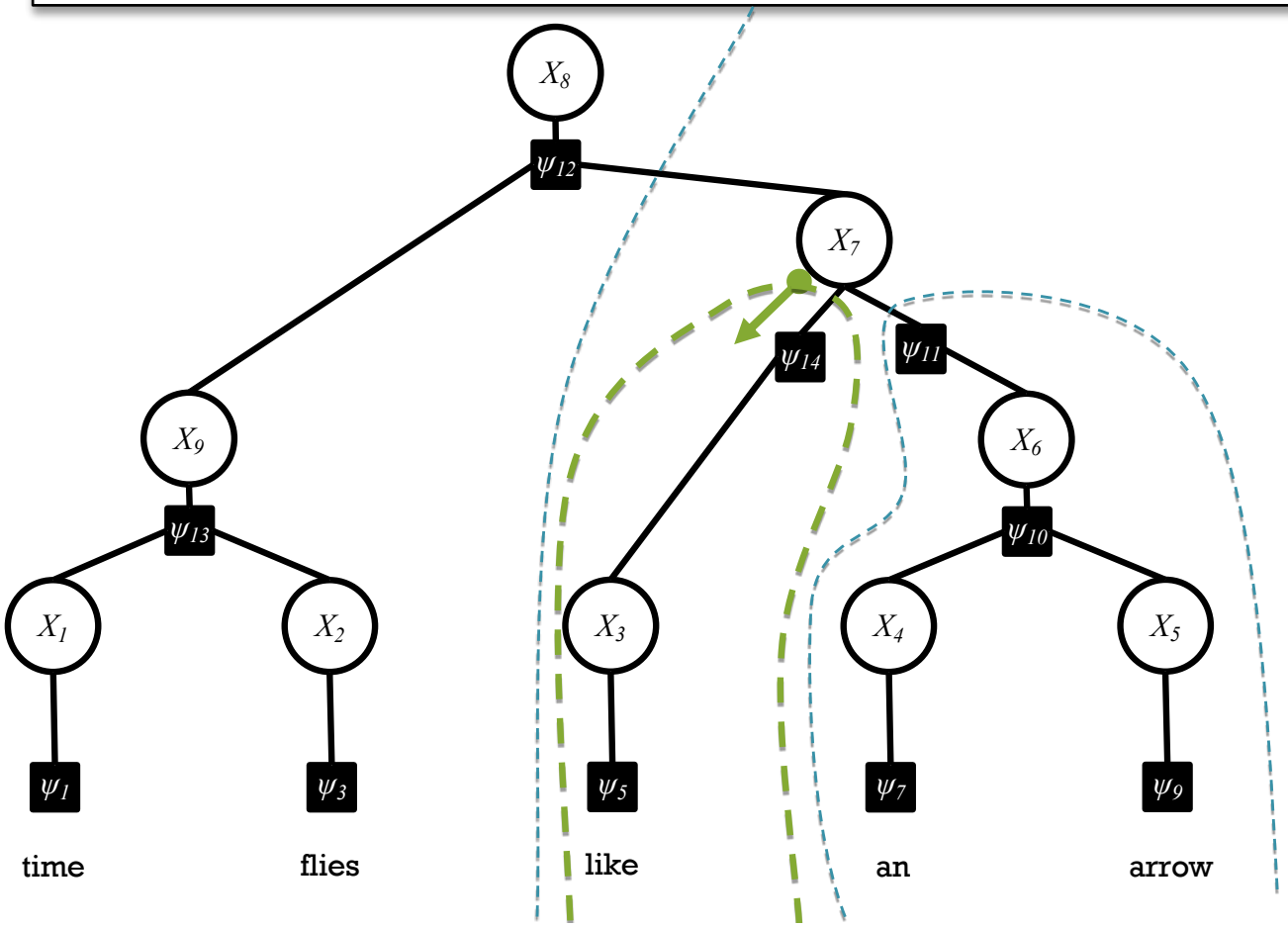
Acyclic BP as Dynamic Programming

- 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.



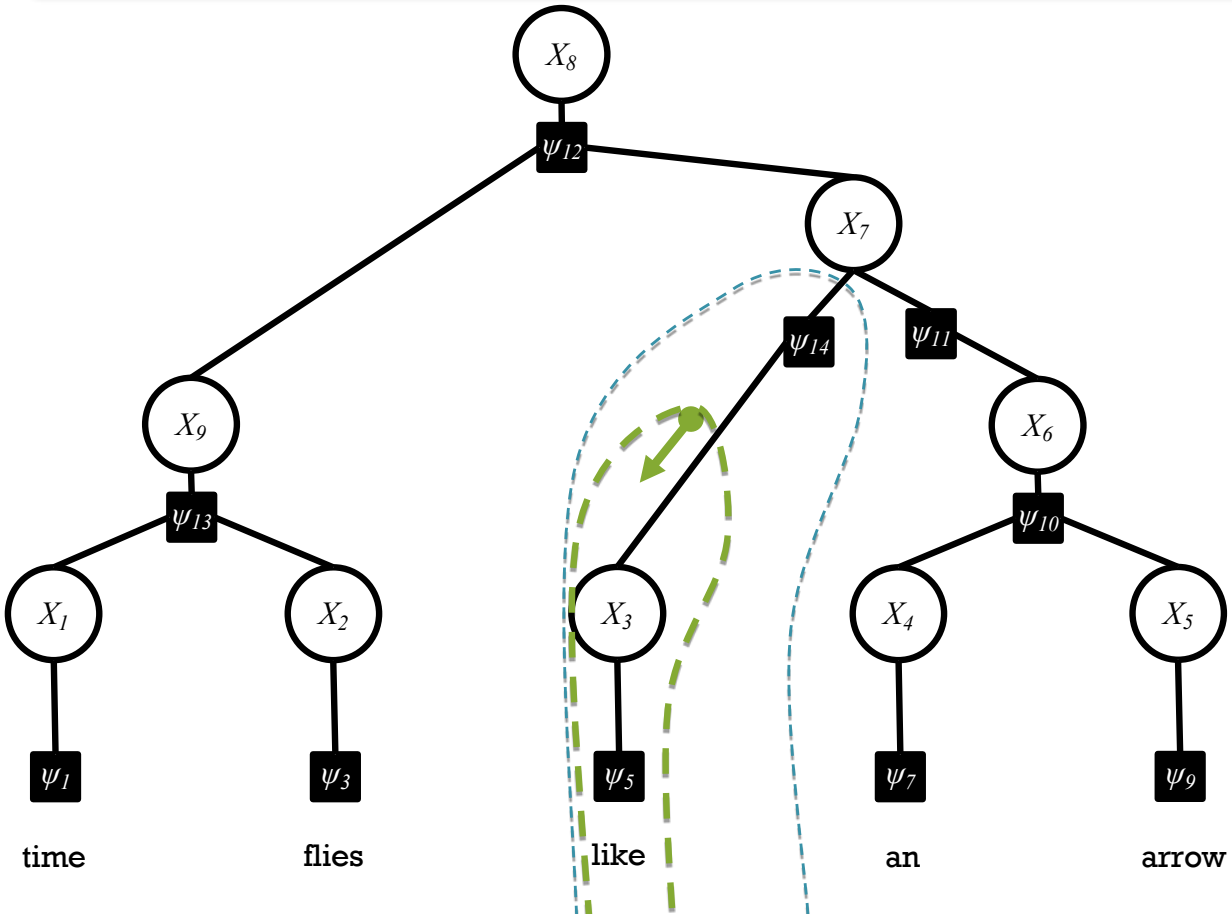
Acyclic BP as Dynamic Programming

- 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.



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,
  $\boldsymbol{X}^* = \text{argmax}_{\boldsymbol{X}}\, p(\boldsymbol{X})$

Max-product Belief Propagation

- Change the sum to a max:

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

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

- **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:

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

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:

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

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

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/+

Exact inference for linear chain models

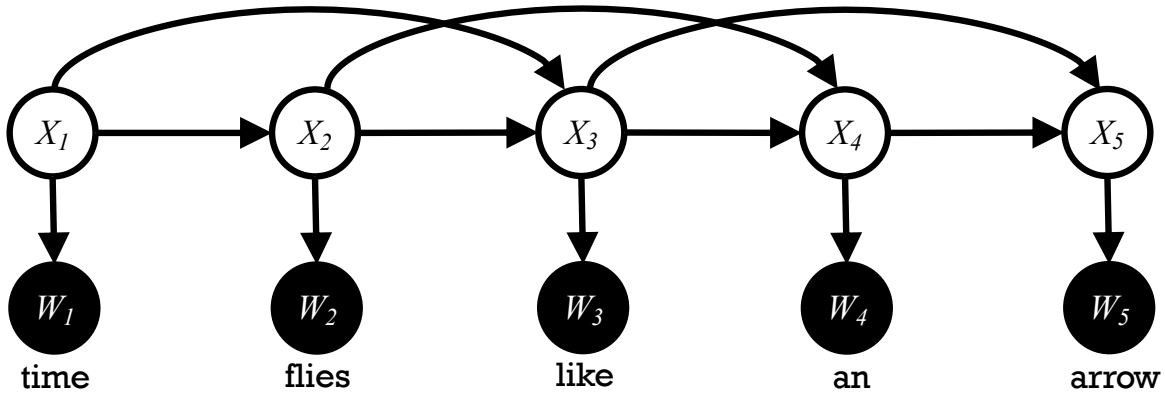
FORWARD-BACKWARD AND VITERBI ALGORITHMS

Forward-Backward Algorithm

- Sum-product BP on an HMM is called the **forward-backward algorithm**
- Max-product BP on an HMM is called the **Viterbi algorithm**

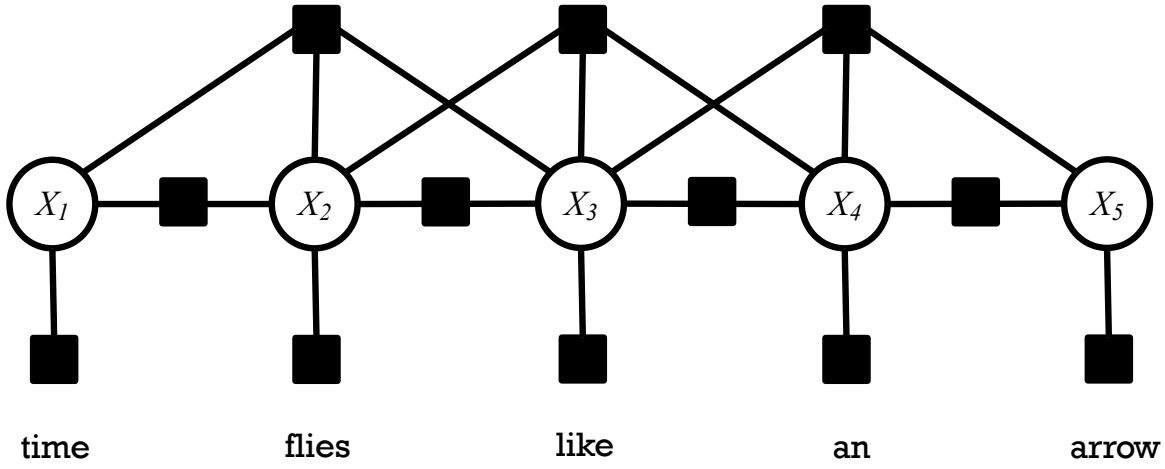
Forward-Backward Algorithm

Trigram HMM is not a tree, even when converted to a factor graph



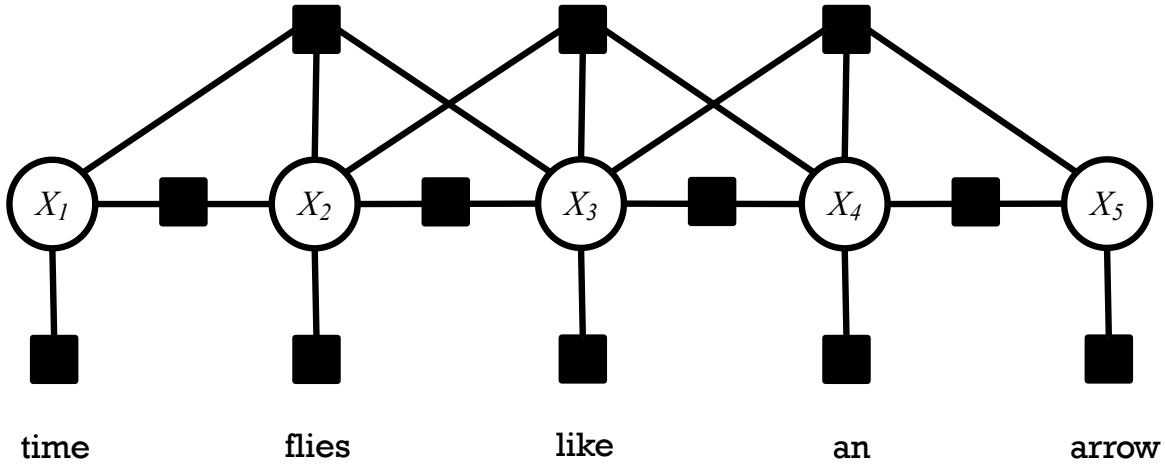
Forward-Backward Algorithm

Trigram HMM is not a tree, even when converted to a factor graph



Forward-Backward Algorithm

Trigram HMM is not a tree, even when converted to a factor graph



Trick: (See also Sha & Pereira (2003))

- Replace each variable domain with its cross product
  e.g. {B,I,O} → {BB, BI, BO, IB, II, IO, OB, OI, OO}
- Replace each pair of variables with a single one. For all i, $y_{i,i+1} = (x_i, x_{i+1})$
- Add features with weight -∞ that disallow illegal configurations between pairs of the new variables
  e.g. **legal** = BI and IO **illegal** = II and OO
- This is effectively a special case of the junction tree algorithm

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.

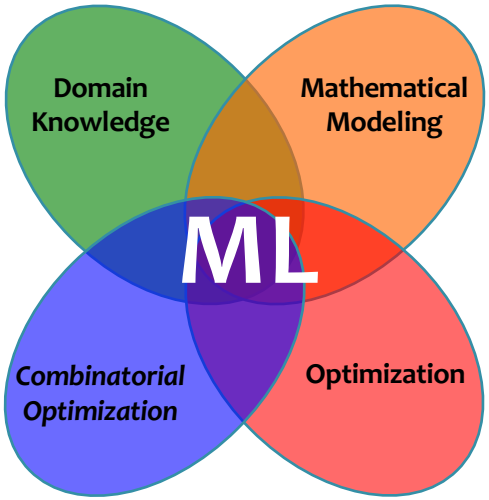
LEARNING FOR MRFS

Machine Learning

The **data** inspires the structures we want to predict

Our **model** defines a score for each structure

It also tells us what to optimize



Inference finds {best structure, marginals, partition function} for a new observation

(**Inference** is usually called as a subroutine in learning)

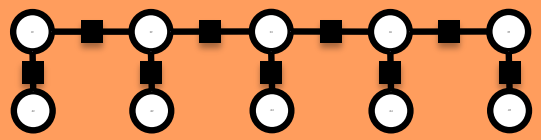
Learning tunes the parameters of the model

1. Data

$$\mathcal{D} = \{\boldsymbol{x}^{(n)}\}_{n=1}^{N}$$

2. Model

$$p(\boldsymbol{x} \mid \boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \prod_{C \in \mathcal{C}} \psi_C(\boldsymbol{x}_C)$$

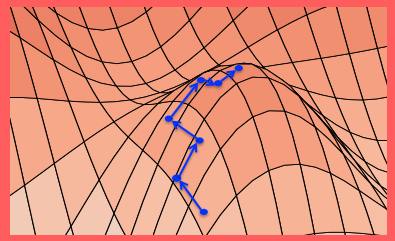


3. Objective

$$\ell(\theta; \mathcal{D}) = \sum_{n=1}^{N} \log p(\boldsymbol{x}^{(n)} \mid \boldsymbol{\theta})$$

4. Learning

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, \ell(\boldsymbol{\theta}; \mathcal{D})$$



5. Inference

1. Marginal Inference

$$p(\boldsymbol{x}_C) = \sum_{\boldsymbol{x}' : \boldsymbol{x}'_C = \boldsymbol{x}_C} p(\boldsymbol{x}' \mid \boldsymbol{\theta})$$

2. Partition Function

$$Z(\boldsymbol{\theta}) = \sum_{\boldsymbol{x}} \prod_{C \in \mathcal{C}} \psi_C(\boldsymbol{x}_C)$$

3. MAP Inference

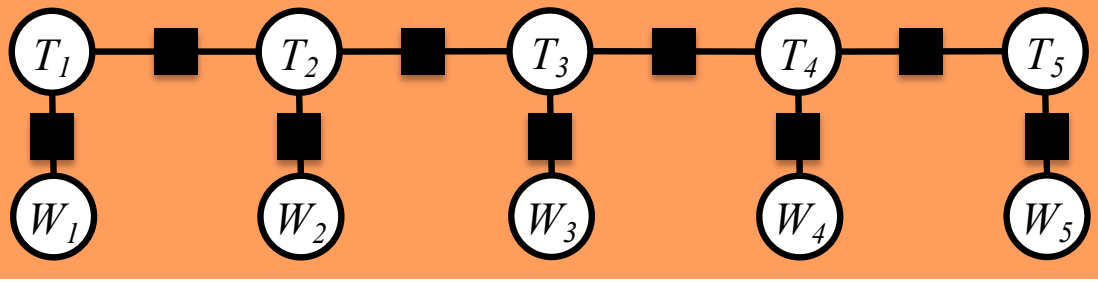
$$\hat{\boldsymbol{x}} = \underset{\boldsymbol{x}}{\operatorname{argmax}}\; p(\boldsymbol{x} \mid \boldsymbol{\theta})$$

1. Data

Given training examples: $\mathcal{D} = \{\boldsymbol{x}^{(n)}\}_{n=1}^{N}$

Sample 1:	n	v	p	d	n
	time	flies	like	an	arrow
Sample 2:	n	n	v	d	n
	time	flies	like	an	arrow
Sample 3:	n	v	p	n	n
	flies	fly	with	their	wings
Sample 4:	p	n	n	v	v
	with	time	you	will	see

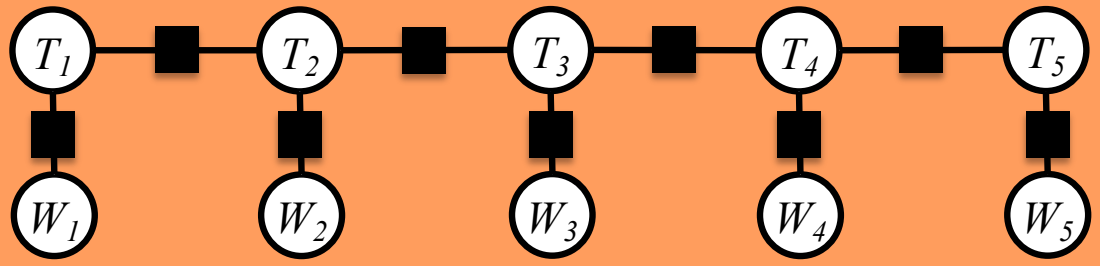
2. Model



2. Model

Define the model to be an MRF:

$$p(\boldsymbol{x} \mid \boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \prod_{C \in \mathcal{C}} \psi_C(\boldsymbol{x}_C)$$



3. Objective

Choose the objective to be log-likelihood:

(Assign high probability to the things we observe and low probability to everything else)

$$\ell(\theta; \mathcal{D}) = \sum_{n=1}^{N} \log p(\boldsymbol{x}^{(n)} \mid \boldsymbol{\theta})$$

3. Objective

Choose the objective to be log-likelihood:

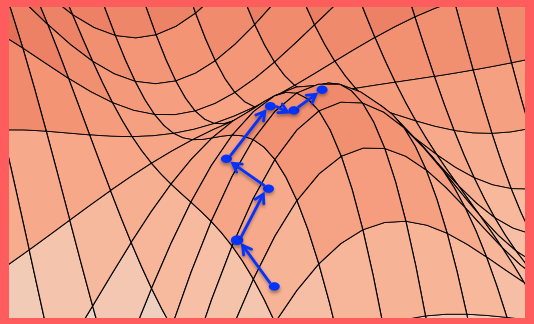
(Assign high probability to the things we observe and low probability to everything else)

$$\ell(\theta; \mathcal{D}) = \sum_{n=1}^{N} \log p(\boldsymbol{x}^{(n)} \mid \boldsymbol{\theta})$$

4. Learning

Tune the parameters to maximize the objective function

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, \ell(\boldsymbol{\theta}; \mathcal{D})$$



3. Objective

Choose the objective to be log-likelihood:

(Assign high probability to the things we observe and low probability to everything else)

$$\ell(\boldsymbol{\theta}; \mathcal{D}) = \sum_{n=1}^{N}$$

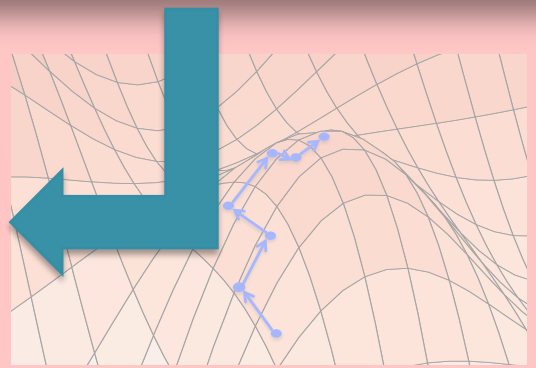
Goals for Today's Lecture

1. Consider different parameterizations
2. Optimize this objective function

Tune the parameter

function

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, \ell(\boldsymbol{\theta}; \mathcal{D})$$



5. Inference

Three Tasks:

1. Marginal Inference

Compute marginals of variables and cliques

$$p(x_i) = \sum_{\boldsymbol{x}' : x'_i = x_i} p(\boldsymbol{x}' \mid \boldsymbol{\theta}) \qquad \Big| \qquad p(\boldsymbol{x}_C) = \sum_{\boldsymbol{x}' : \boldsymbol{x}'_C = \boldsymbol{x}_C} p(\boldsymbol{x}' \mid \boldsymbol{\theta})$$

2. Partition Function

Compute the normalization constant

$$Z(\boldsymbol{\theta}) = \sum_{\boldsymbol{x}} \prod_{C \in \mathcal{C}} \psi_C(\boldsymbol{x}_C)$$

3. MAP Inference

Compute variable assignment with highest probability

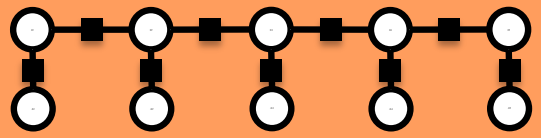
$$\hat{\boldsymbol{x}} = \operatorname*{argmax}_{\boldsymbol{x}} \; p(\boldsymbol{x} \mid \boldsymbol{\theta})$$

1. Data

$$\mathcal{D} = \{\boldsymbol{x}^{(n)}\}_{n=1}^{N}$$

2. Model

$$p(\boldsymbol{x} \mid \boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \prod_{C \in \mathcal{C}} \psi_C(\boldsymbol{x}_C)$$

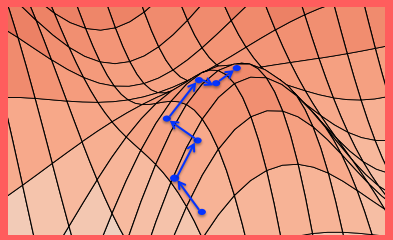


3. Objective

$$\ell(\theta; \mathcal{D}) = \sum_{n=1}^{N} \log p(\boldsymbol{x}^{(n)} \mid \boldsymbol{\theta})$$

4. Learning

$$\boldsymbol{\theta}^* = \operatorname*{argmax}_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}; \mathcal{D})$$



5. Inference

1. Marginal Inference

$$p(\boldsymbol{x}_C) = \sum_{\boldsymbol{x}' : \boldsymbol{x}'_C = \boldsymbol{x}_C} p(\boldsymbol{x}' \mid \boldsymbol{\theta})$$

2. Partition Function

$$Z(\boldsymbol{\theta}) = \sum_{\boldsymbol{x}} \prod_{C \in \mathcal{C}} \psi_C(\boldsymbol{x}_C)$$

3. MAP Inference

$$\hat{\boldsymbol{x}} = \operatorname*{argmax}_{\boldsymbol{x}} p(\boldsymbol{x} \mid \boldsymbol{\theta})$$

MLE for Undirected GMs

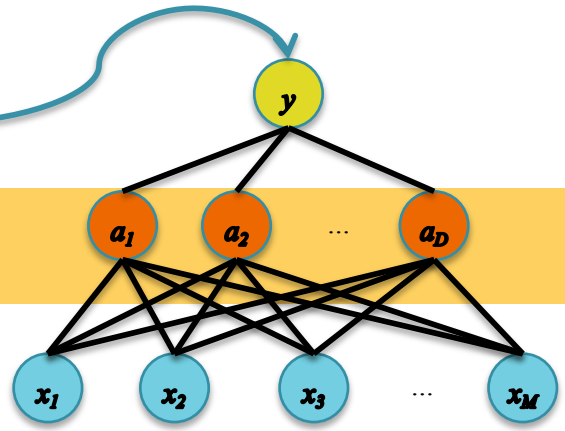
- Today's parameter estimation assumptions:
  1. The graphical model structure is given
  2. Every variable appears in the training examples

Questions

1. What does the **likelihood objective** accomplish?
2. Is likelihood the ***right* objective** function?
3. **How do we optimize** the objective function (i.e. learn)?
4. What **guarantees** does the optimizer provide?
5. (What is the **mapping from data → model**? In what ways can we incorporate our domain knowledge? How does this impact learning?)

Options for MLE of MRFs

- **Setting I:** $\psi_C(\boldsymbol{x}_C) = \theta_{C,\boldsymbol{x}_C}$
  - A. MLE by inspection (Decomposable Models)
  - B. Iterative Proportional Fitting (IPF)
- **Setting II:** $\psi_C(\boldsymbol{x}_C) = \exp(\boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}_C))$
  - C. Generalized Iterative Scaling
  - D. Gradient-based Methods
- **Setting III:** $\psi_C(\boldsymbol{x}_C) =$
  - E. Gradient-based Methods



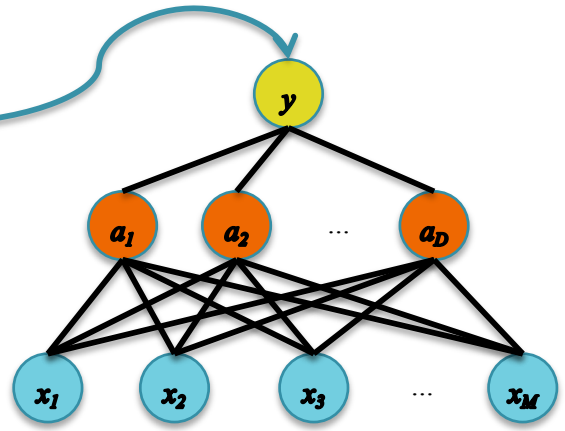
MRF LEARNING (TRIVIAL CASE)

Options for MLE of MRFs

- **Setting I:** $\psi_C(\boldsymbol{x}_C) = \theta_{C,\boldsymbol{x}_C}$
  A. MLE by inspection (Decomposable Models)
  B. Iterative Proportional Fitting (IPF)

- **Setting II:** $\psi_C(\boldsymbol{x}_C) = \exp(\boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}_C))$
  C. Generalized Iterative Scaling
  D. Gradient-based Methods

- **Setting III:** $\psi_C(\boldsymbol{x}_C) =$
  E. Gradient-based Methods



MLE by Inspection

Whiteboard:

- Example 1: linear-chain on three variables
- Example 2: "decomposable" with four variables

MLE by Inspection

- **Definition:** Graph is **decomposable** if it can be recursively subdivided into sets A, B, and S such that S separates A and B.

MLE by Inspection

- **Definition:** Graph is **decomposable** if it can be recursively subdivided into sets A, B, and S such that S separates A and B.
- **Recipe for MLE by Guessing:**
  - Three conditions:
    1. Graphical model is *decomposable*
    2. Potentials defined on *maximal cliques*
    3. Potentials are are parameterized as: $\psi_C(\boldsymbol{x}_C) = \theta_{C,\boldsymbol{x}_C}$
  - **Step 1:** set each clique potential to its empirical marginal
  - **Step 2:** divide out every non-empty intersection between cliques *exactly once*

MLE by Inspection

- **Definition:** Graph is **decomposable** if it can be recursively subdivided into sets A, B, and S such that S separates A and B.
- **Recipe for MLE by Guessing:**
  - Three conditions:
    1. Graphical model is *decomposable*
    2. Potentials defined on *maximal cliques*
    3. Potentials are are parameterized as: $\psi_C(\boldsymbol{x}_C) = \theta_{C,\boldsymbol{x}_C}$
  - **Step 1:** set each clique potential to its empirical marginal
  - **Step 2:** divide out every non-empty intersection between cliques *exactly once*

How is this different than learning tabular Bayesian Networks?

MRF LEARNING (LOG-LINEAR CASE)

Options for MLE of MRFs

- **Setting I:** $\psi_C(\boldsymbol{x}_C) = \theta_{C,\boldsymbol{x}_C}$
  A. MLE by inspection (Decomposable Models)
  B. Iterative Proportional Fitting (IPF)

- **Setting II:** $\psi_C(\boldsymbol{x}_C) = \exp(\boldsymbol{\theta} \cdot \boldsymbol{f}(\boldsymbol{x}_C))$
  C. Generalized Iterative Scaling
  D. Gradient-based Methods

- **Setting III:** $\psi_C(\boldsymbol{x}_C) =$
  E. Gradient-based Methods