Structured Belief Propagation for NLP

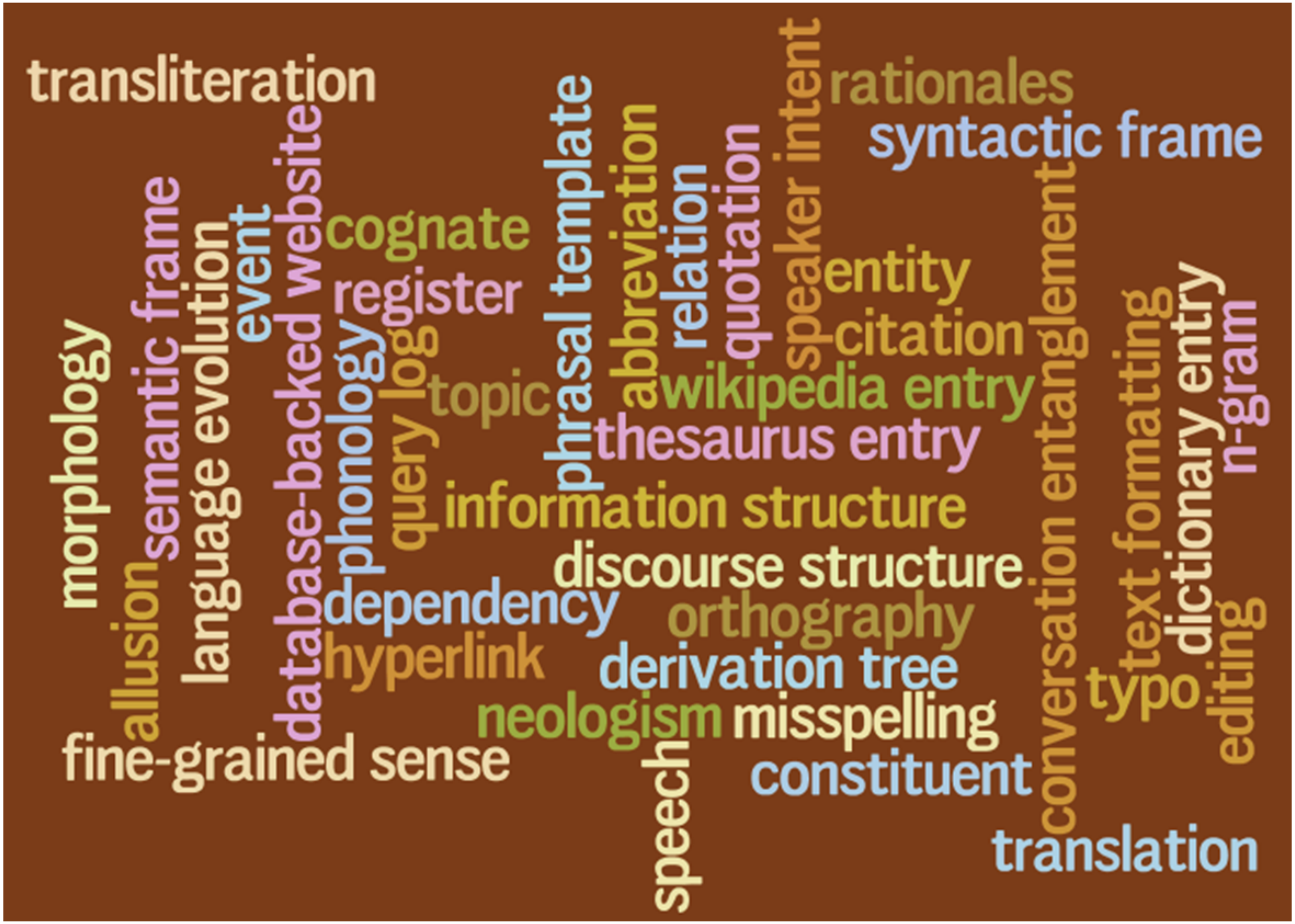
Matthew R. Gormley & Jason Eisner

ACL '15 Tutorial

July 26, 2015

For the latest version of these slides, please visit:
http://www.cs.jhu.edu/~mrg/bp-tutorial/

Language has a lot going on at once



Structured representations of utterances
Structured knowledge of the language

Many interacting parts for BP to reason about!

Outline

- Do you want to push past the simple NLP models (logistic regression, PCFG, etc.) that we've all been using for 20 years?
- Then this tutorial is extremely practical for you!
  1. **Models:** Factor graphs can express interactions among linguistic structures.
  2. **Algorithm:** BP estimates the global effect of these interactions on each variable, using local computations.
  3. **Intuitions:** What's going on here? Can we trust BP's estimates?
  4. **Fancier Models:** Hide a whole grammar and dynamic programming algorithm within a single factor. BP coordinates multiple factors.
  5. **Tweaked Algorithm:** Finish in fewer steps and make the steps faster.
  6. **Learning:** Tune the parameters. Approximately improve the true predictions -- or truly improve the approximate predictions.
  7. **Software:** Build the model you want!

Outline

- Do you want to push past the simple NLP models (logistic regression, PCFG, etc.) that we've all been using for 20 years?
- Then this tutorial is extremely practical for you!

1. **Models:** Factor graphs can express interactions among linguistic structures.
2. **Algorithm:** BP estimates the global effect of these interactions on each variable, using local computations.
3. **Intuitions:** What's going on here? Can we trust BP's estimates?
4. **Fancier Models:** Hide a whole grammar and dynamic programming algorithm within a single factor. BP coordinates multiple factors.
5. **Tweaked Algorithm:** Finish in fewer steps and make the steps faster.
6. **Learning:** Tune the parameters. Approximately improve the true predictions -- or truly improve the approximate predictions.
7. **Software:** Build the model you want!

Outline

- Do you want to push past the simple NLP models (logistic regression, PCFG, etc.) that we've all been using for 20 years?
- Then this tutorial is extremely practical for you!
  1. **Models:** Factor graphs can express interactions among linguistic structures.
  2. **Algorithm:** BP estimates the global effect of these interactions on each variable, using local computations.
  3. **Intuitions:** What's going on here? Can we trust BP's estimates?
  4. **Fancier Models:** Hide a whole grammar and dynamic programming algorithm within a single factor. BP coordinates multiple factors.
  5. **Tweaked Algorithm:** Finish in fewer steps and make the steps faster.
  6. **Learning:** Tune the parameters. Approximately improve the true predictions -- or truly improve the approximate predictions.
  7. **Software:** Build the model you want!

Outline

- Do you want to push past the simple NLP models (logistic regression, PCFG, etc.) that we've all been using for 20 years?
- Then this tutorial is extremely practical for you!
  1. **Models:** Factor graphs can express interactions among linguistic structures.
  2. **Algorithm:** BP estimates the global effect of these interactions on each variable, using local computations.
  3. **Intuitions:** What's going on here? Can we trust BP's estimates?
  4. **Fancier Models:** Hide a whole grammar and dynamic programming algorithm within a single factor. BP coordinates multiple factors.
  5. **Tweaked Algorithm:** Finish in fewer steps and make the steps faster.
  6. **Learning:** Tune the parameters. Approximately improve the true predictions -- or truly improve the approximate predictions.
  7. **Software:** Build the model you want!

Outline

- Do you want to push past the simple NLP models (logistic regression, PCFG, etc.) that we've all been using for 20 years?
- Then this tutorial is extremely practical for you!
  1. **Models:** Factor graphs can express interactions among linguistic structures.
  2. **Algorithm:** BP estimates the global effect of these interactions on each variable, using local computations.
  3. **Intuitions:** What's going on here? Can we trust BP's estimates?
  4. **Fancier Models:** Hide a whole grammar and dynamic programming algorithm within a single factor. BP coordinates multiple factors.
  5. **Tweaked Algorithm:** Finish in fewer steps and make the steps faster.
  6. **Learning:** Tune the parameters. Approximately improve the true predictions -- or truly improve the approximate predictions.
  7. **Software:** Build the model you want!

Outline

- Do you want to push past the simple NLP models (logistic regression, PCFG, etc.) that we've all been using for 20 years?
- Then this tutorial is extremely practical for you!

1. **Models:** Factor graphs can express interactions among linguistic structures.
2. **Algorithm:** BP estimates the global effect of these interactions on each variable, using local computations.
3. **Intuitions:** What's going on here? Can we trust BP's estimates?
4. **Fancier Models:** Hide a whole grammar and dynamic programming algorithm within a single factor. BP coordinates multiple factors.
5. **Tweaked Algorithm:** Finish in fewer steps and make the steps faster.
6. **Learning:** Tune the parameters. Approximately improve the true predictions -- or truly improve the approximate predictions.
7. **Software:** Build the model you want!

Outline

- Do you want to push past the simple NLP models (logistic regression, PCFG, etc.) that we've all been using for 20 years?
- Then this tutorial is extremely practical for you!
  1. **Models:** Factor graphs can express interactions among linguistic structures.
  2. **Algorithm:** BP estimates the global effect of these interactions on each variable, using local computations.
  3. **Intuitions:** What's going on here? Can we trust BP's estimates?
  4. **Fancier Models:** Hide a whole grammar and dynamic programming algorithm within a single factor. BP coordinates multiple factors.
  5. **Tweaked Algorithm:** Finish in fewer steps and make the steps faster.
  6. **Learning:** Tune the parameters. Approximately improve the true predictions -- or truly improve the approximate predictions.
  7. **Software:** Build the model you want!

Outline

- Do you want to push past the simple NLP models (logistic regression, PCFG, etc.) that we've all been using for 20 years?
- Then this tutorial is extremely practical for you!

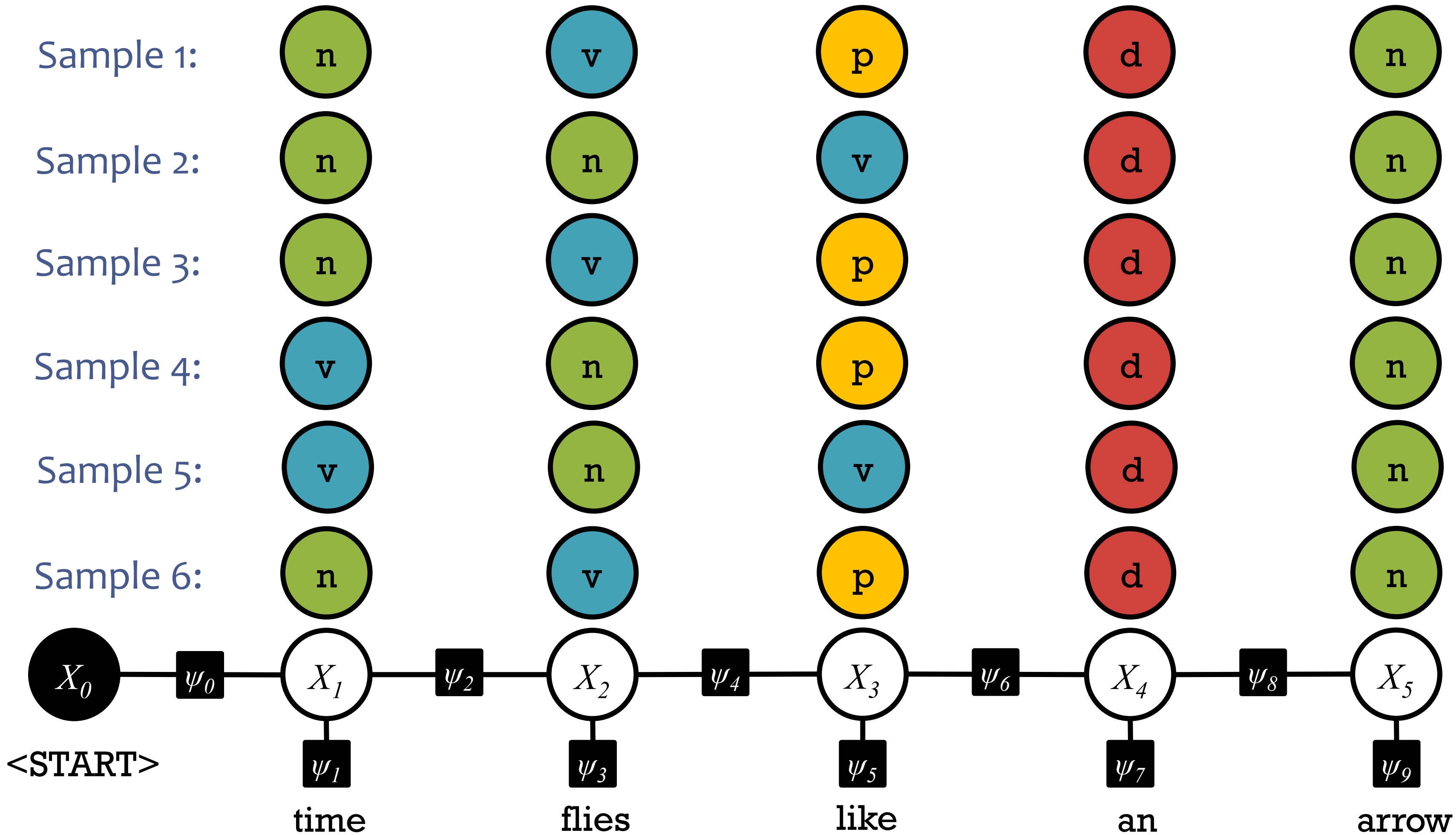
1. **Models:** Factor graphs can express interactions among linguistic structures.
2. **Algorithm:** BP estimates the global effect of these interactions on each variable, using local computations.
3. **Intuitions:** What's going on here? Can we trust BP's estimates?
4. **Fancier Models:** Hide a whole grammar and dynamic programming algorithm within a single factor. BP coordinates multiple factors.
5. **Tweaked Algorithm:** Finish in fewer steps and make the steps faster.
6. **Learning:** Tune the parameters. Approximately improve the true predictions -- or truly improve the approximate predictions.
7. **Software:** Build the model you want!

Section 1: Introduction

Modeling with Factor Graphs

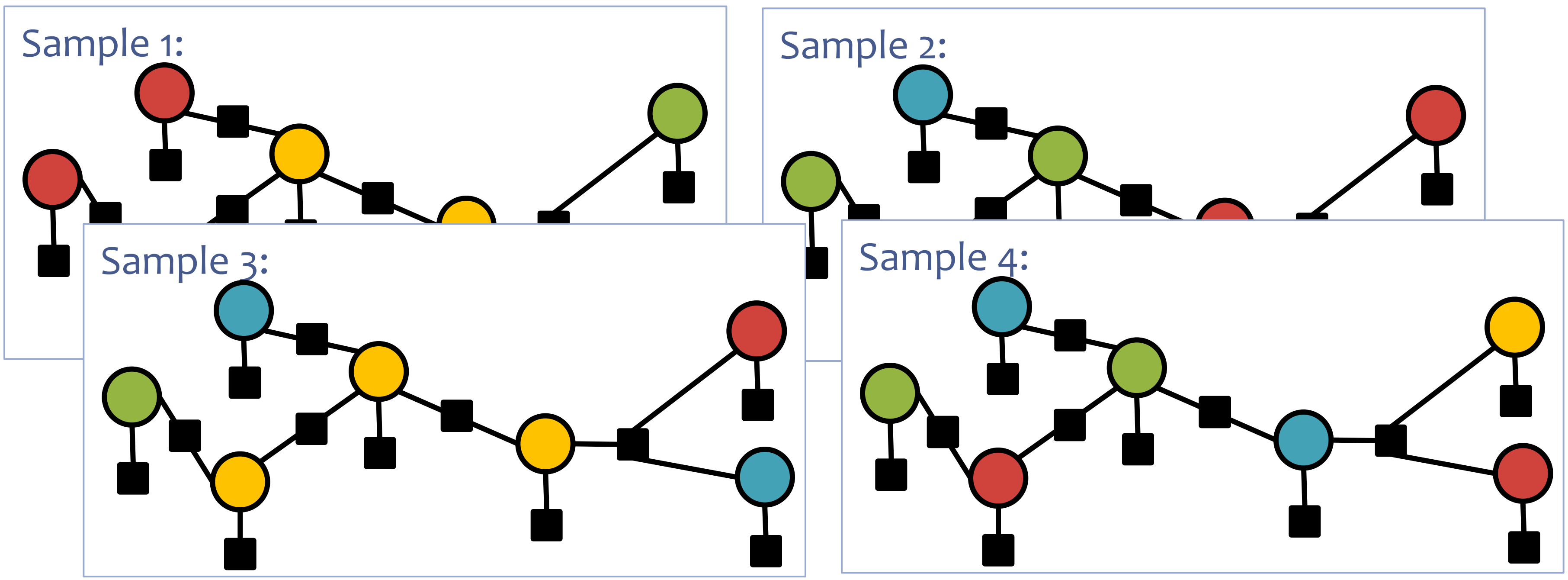
Sampling from a Joint Distribution

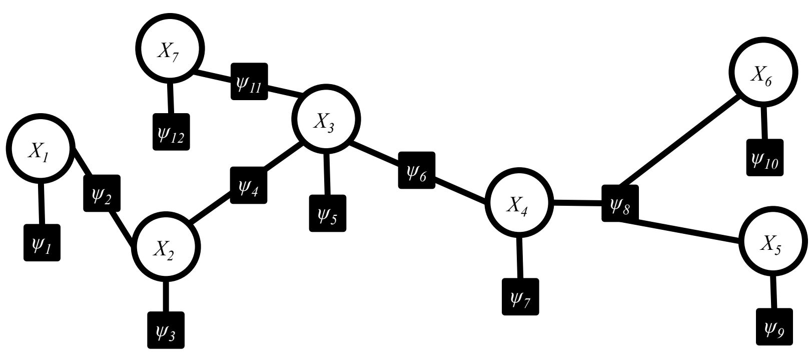
A **joint distribution** defines a probability $p(\boldsymbol{x})$ for each assignment of values $\boldsymbol{x}$ to variables $\boldsymbol{X}$. This gives the **proportion** of samples that will equal $\boldsymbol{x}$.



Sampling from a Joint Distribution

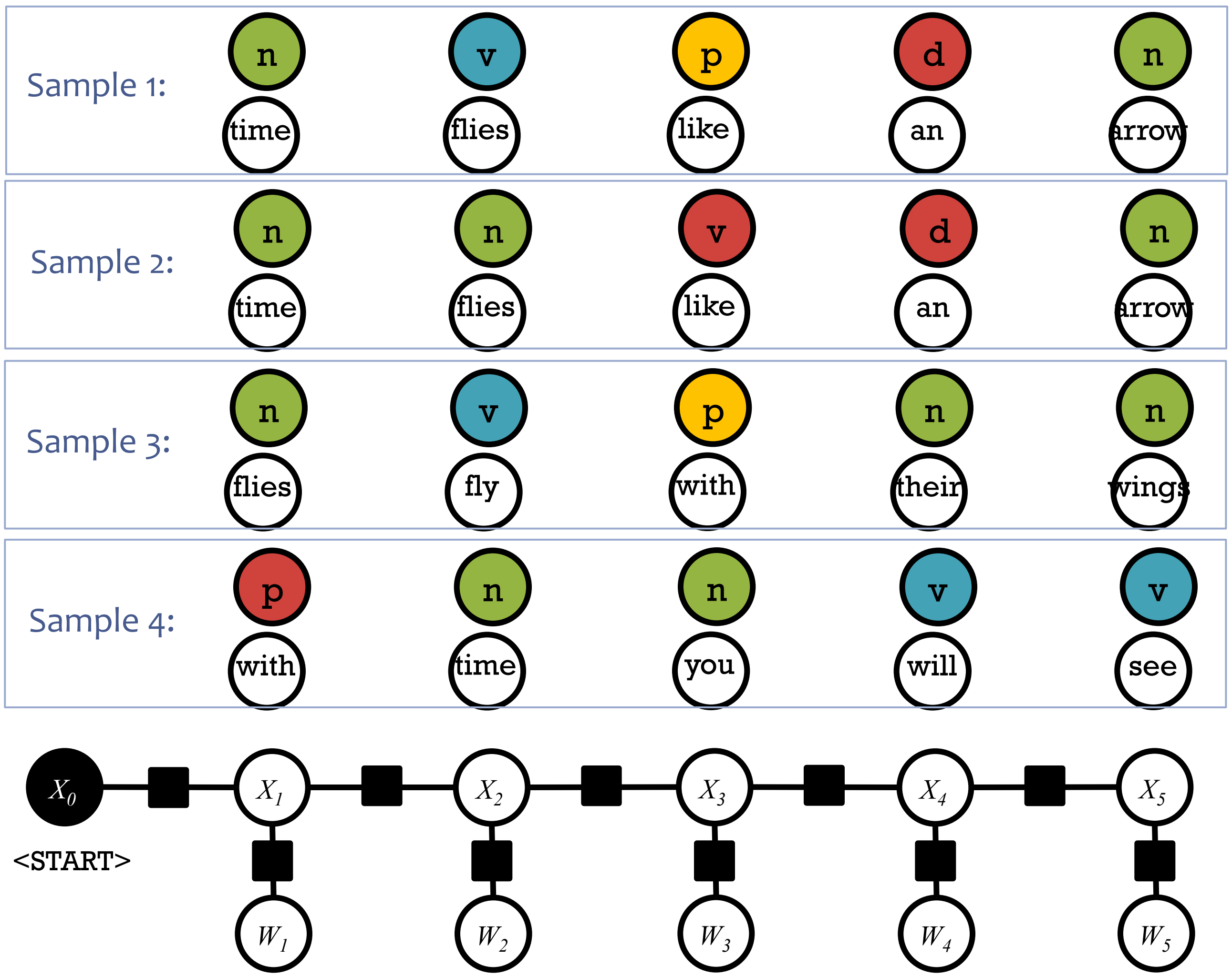
A **joint distribution** defines a probability $p(\boldsymbol{x})$ for each assignment of values $\boldsymbol{x}$ to variables $\boldsymbol{X}$.
This gives the **proportion** of samples that will equal $\boldsymbol{x}$.





Sampling from a Joint Distribution

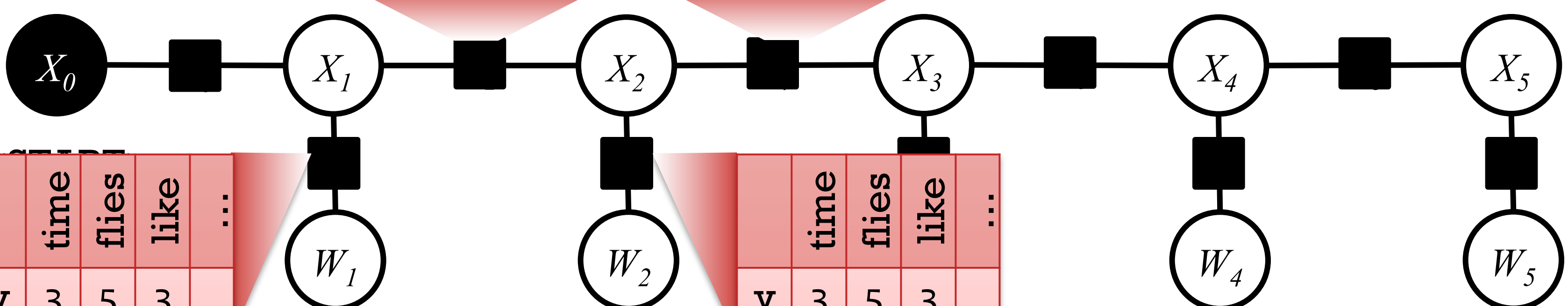
A **joint distribution** defines a probability $p(\boldsymbol{x})$ for each assignment of values $\boldsymbol{x}$ to variables $\boldsymbol{X}$. This gives the **proportion** of samples that will equal $\boldsymbol{x}$.



Factors have local opinions (≥ 0)

Each black box looks at *some* of the tags X_i and words W_i

Note: We chose to reuse the same factors at different positions in the sentence.



	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0

	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0

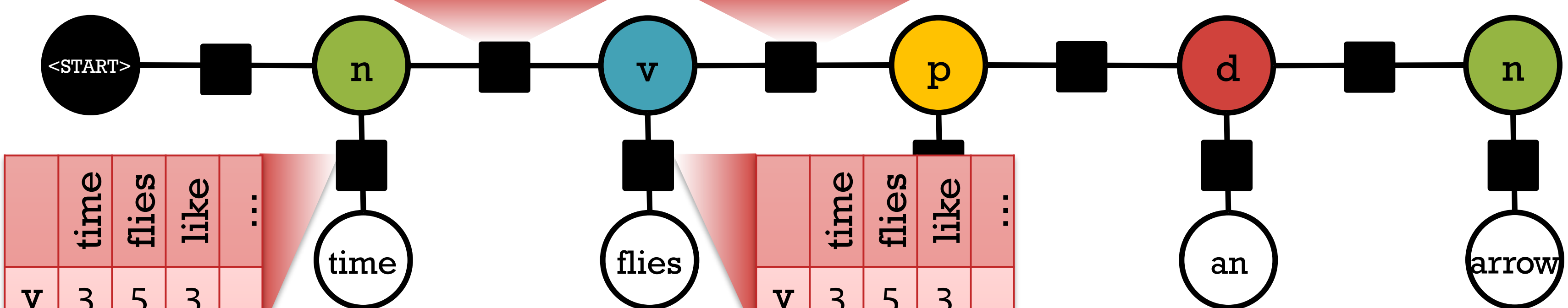
	time	flies	like	…
v	3	5	3	
n	4	5	2	
p	0.1	0.1	3	
d	0.1	0.2	0.1	

	time	flies	like	…
v	3	5	3	
n	4	5	2	
p	0.1	0.1	3	
d	0.1	0.2	0.1	

Factors have local opinions (≥ 0)

Each black box looks at *some* of the tags X_i and words W_i

$$p(\text{n, v, p, d, n, time, flies, like, an, arrow}) \quad = \quad ?$$



	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0

	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0

	time	flies	like	…
v	3	5	3	
n	4	5	2	
p	0.1	0.1	3	
d	0.1	0.2	0.1	

	time	flies	like	…
v	3	5	3	
n	4	5	2	
p	0.1	0.1	3	
d	0.1	0.2	0.1	

Global probability = product of local opinions

Each black box looks at *some* of the tags X_i and words W_i

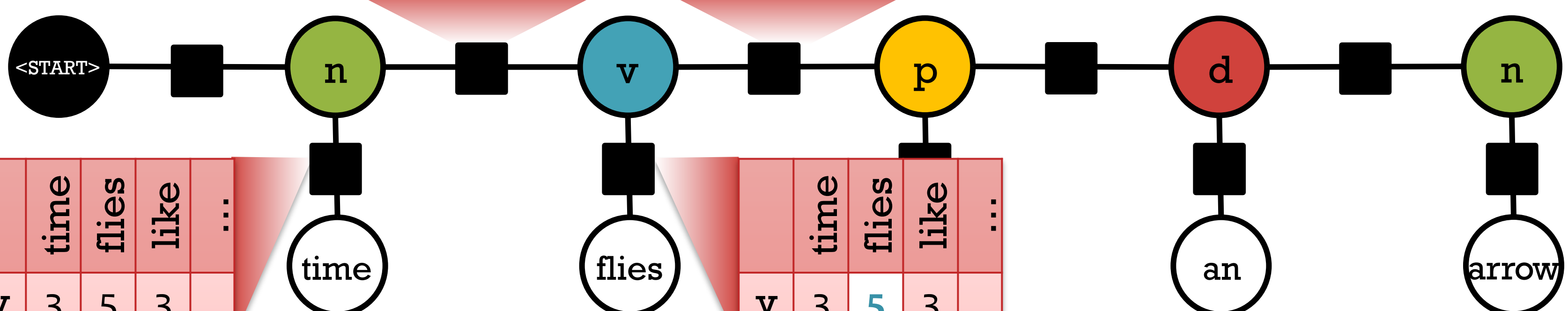
$$p(\text{n, v, p, d, n, time, flies, like, an, arrow}) = \frac{1}{Z}(4 * 8 * 5 * 3 * \ldots)$$

	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0

	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0

Uh-oh! The probabilities of the various assignments sum up to Z > 1.

So divide them all by Z.



	time	flies	like	…
v	3	5	3	
n	4	5	2	
p	0.1	0.1	3	
d	0.1	0.2	0.1	

	time	flies	like	…
v	3	5	3	
n	4	5	2	
p	0.1	0.1	3	
d	0.1	0.2	0.1	

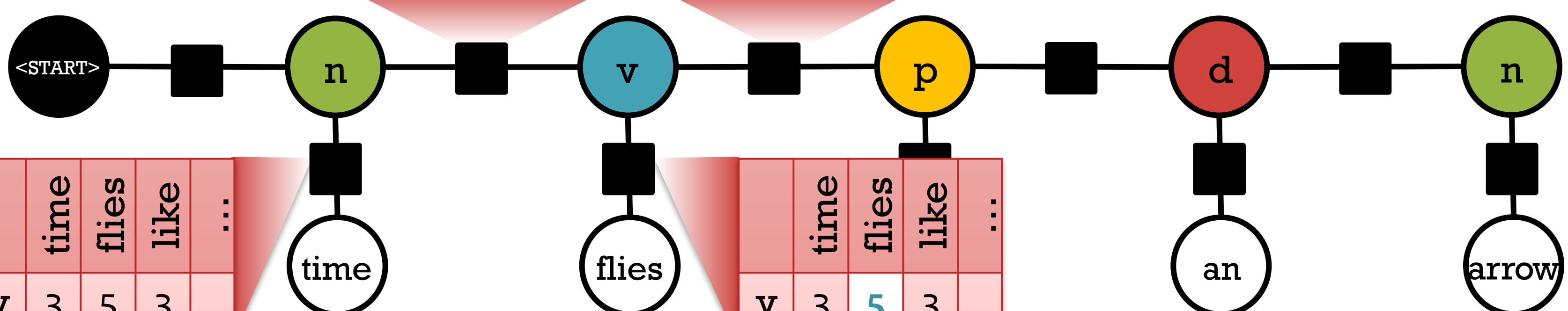
Markov Random Field (MRF)

Joint distribution over tags X_i and words W_i
The individual factors aren't *necessarily* probabilities.

$$p(\text{n, v, p, d, n, time, flies, like, an, arrow}) = \frac{1}{Z}(4 * 8 * 5 * 3 * \ldots)$$

	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0

	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0



	time	flies	like	…
v	3	5	3	
n	4	5	2	
p	0.1	0.1	3	
d	0.1	0.2	0.1	

	time	flies	like	…
v	3	5	3	
n	4	5	2	
p	0.1	0.1	3	
d	0.1	0.2	0.1	

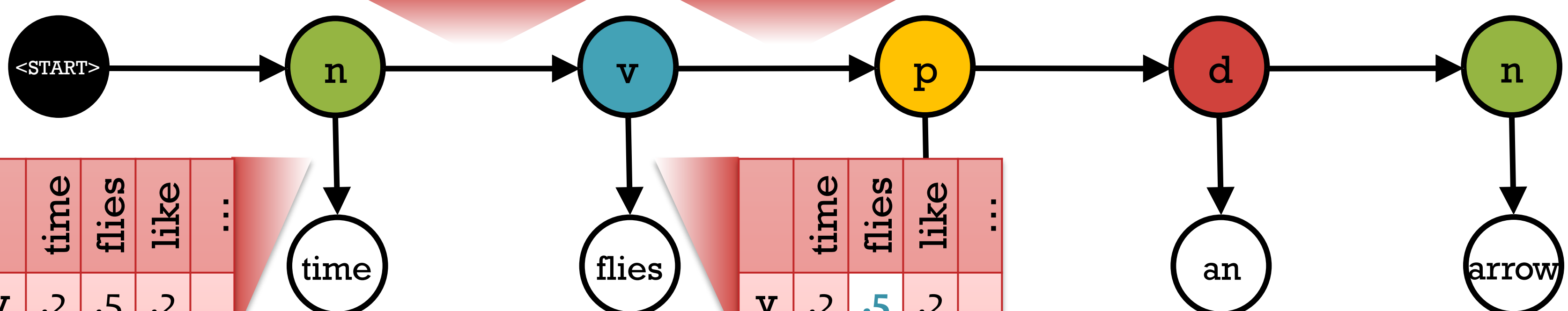
Hidden Markov Model

But sometimes we *choose* to make them probabilities.
Constrain each row of a factor to sum to one. Now $Z = 1$.

$$p(\text{n, v, p, d, n, time, flies, like, an, arrow}) = \cancel{\frac{1}{Z}}(.3 * .8 * .2 * .5 * \ldots)$$

	v	n	p	d
v	.1	.4	.2	.3
n	.8	.1	.1	0
p	.2	.3	.2	.3
d	.2	.8	0	0

	v	n	p	d
v	.1	.4	.2	.3
n	.8	.1	.1	0
p	.2	.3	.2	.3
d	.2	.8	0	0



	time	flies	like	…
v	.2	.5	.2	
n	.3	.4	.2	
p	.1	.1	.3	
d	.1	.2	.1	

	time	flies	like	…
v	.2	.5	.2	
n	.3	.4	.2	
p	.1	.1	.3	
d	.1	.2	.1	

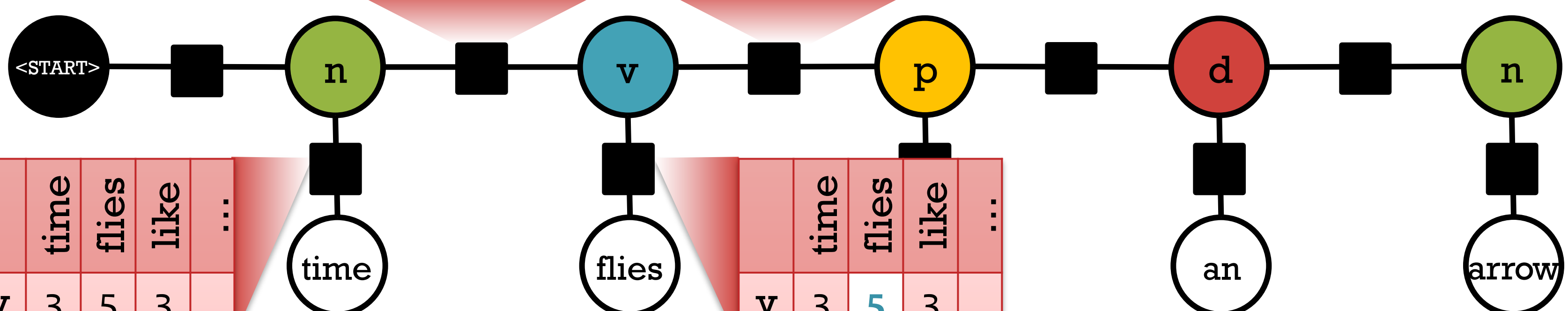
Markov Random Field (MRF)

Joint distribution over tags X_i and words W_i

$$p(\text{n, v, p, d, n, time, flies, like, an, arrow}) = \frac{1}{Z}(4 * 8 * 5 * 3 * \ldots)$$

	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0

	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0



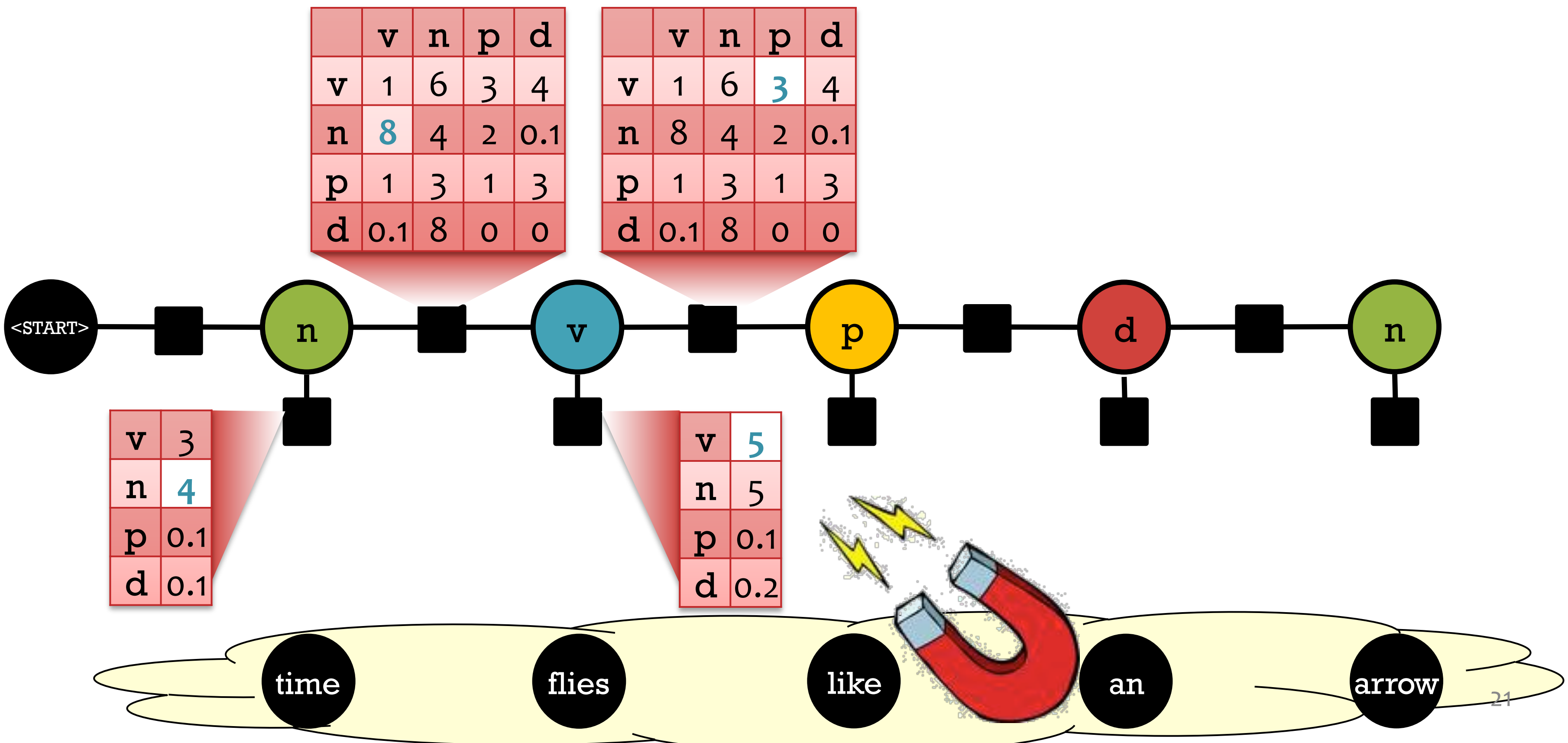
	time	flies	like	…
v	3	5	3	
n	4	5	2	
p	0.1	0.1	3	
d	0.1	0.2	0.1	

	time	flies	like	…
v	3	5	3	
n	4	5	2	
p	0.1	0.1	3	
d	0.1	0.2	0.1	

Conditional Random Field (CRF)

Conditional distribution over tags X_i <u>given</u> words w_i.
The factors and Z are now specific to the sentence w.

$$p(\text{n, v, p, d, n, time, flies, like, an, arrow}) = \frac{1}{Z}(4 * 8 * 5 * 3 * \ldots)$$



	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0

	v	n	p	d
v	1	6	3	4
n	8	4	2	0.1
p	1	3	1	3
d	0.1	8	0	0

v	3
n	4
p	0.1
d	0.1

v	5
n	5
p	0.1
d	0.2

How General Are Factor Graphs?

- Factor graphs can be used to describe
  - Markov Random Fields (undirected graphical models)
    - i.e., log-linear models over a tuple of variables
  - Conditional Random Fields
  - Bayesian Networks (directed graphical models)

- *Inference* treats all of these interchangeably.
  - Convert your model to a factor graph first.
  - Pearl (1988) gave key strategies for *exact* inference:
    - **Belief propagation**, for inference on *acyclic* graphs
    - **Junction tree algorithm**, for making *any* graph acyclic (by merging variables and factors: blows up the runtime)

Object-Oriented Analogy

- **What is a sample?**
  A datum: an immutable *object* that describes a linguistic structure.
- **What is the sample space?**
  The *class* of all possible sample objects.

```
class Tagging:

    int n;               // length of sentence
    Word[] w;            // array of n words (values w_i)
    Tag[] t;             // array of n tags  (values t_i)
```

- **What is a random variable?**
  An *accessor method* of the class, e.g., one that returns a certain field.
  - Will give different values when called on different random samples.

```
Word W(int i) { return w[i]; }     // random var W_i
Tag T(int i) { return t[i]; }      // random var T_i

String S(int i) {                  // random var S_i
    return suffix(w[i], 3);  }
```

Random variable W_5 takes value w_5 == "arrow" in this sample

Object-Oriented Analogy

- **What is a sample?**
  A datum: an immutable *object* that describes a linguistic structure.
- **What is the sample space?**
  The *class* of all possible sample objects.
- **What is a random variable?**
  An *accessor method* of the class, e.g., one that returns a certain field.

- **A model is represented by a different object. What is a factor of the model?**
  A *method* of the model that computes a number ≥ 0 from a sample, based on the sample's values of a few random variables, and parameters stored in the model.

```
class TaggingModel:
    float transition(Tagging tagging, int i) {   // tag-tag bigram
        return tparam[tagging.t(i-1)][tagging.t(i)]; }
    float emission(Tagging tagging, int i) {     // tag-word bigram
        return eparam[tagging.t(i)][tagging.w(i)]; }
```

- **How do you find the scaling factor?**
  Add up the probabilities of all *possible* samples. If the result Z != 1, divide the probabilities by that Z.

```
    float uprob(Tagging tagging) {    // unnormalized prob
        float p=1;
        for (i=1; i <= tagging.n; i++) {
            p *= transition(i) * emission(i); }    return p; }
```

Modeling with Factor Graphs

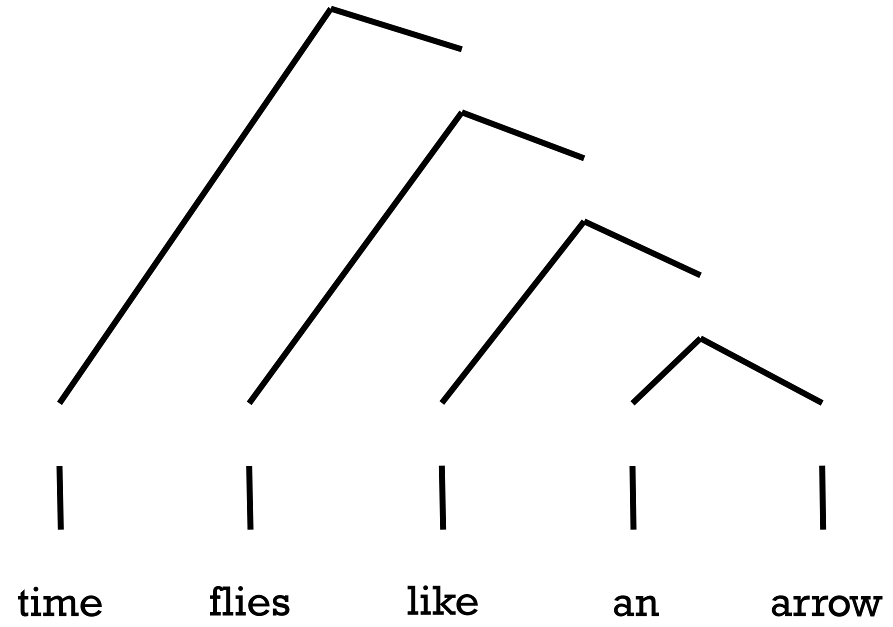
- **Factor graphs** can be used to model many **linguistic structures.**

- Here we highlight a few **example NLP tasks.**
  - *People have used BP for all of these.*

- We'll describe how **variables** and **factors** were used to describe structures and the interactions among their parts.

Annotating a Tree

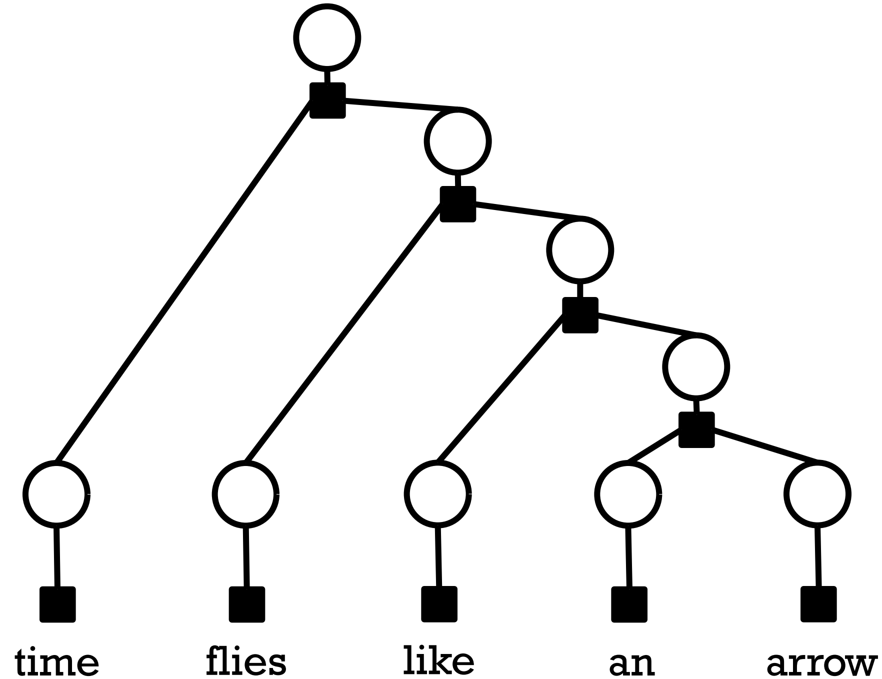
Given: a **sentence** and unlabeled **parse** tree.



Annotating a Tree

Given: a **sentence** and unlabeled **parse** tree.

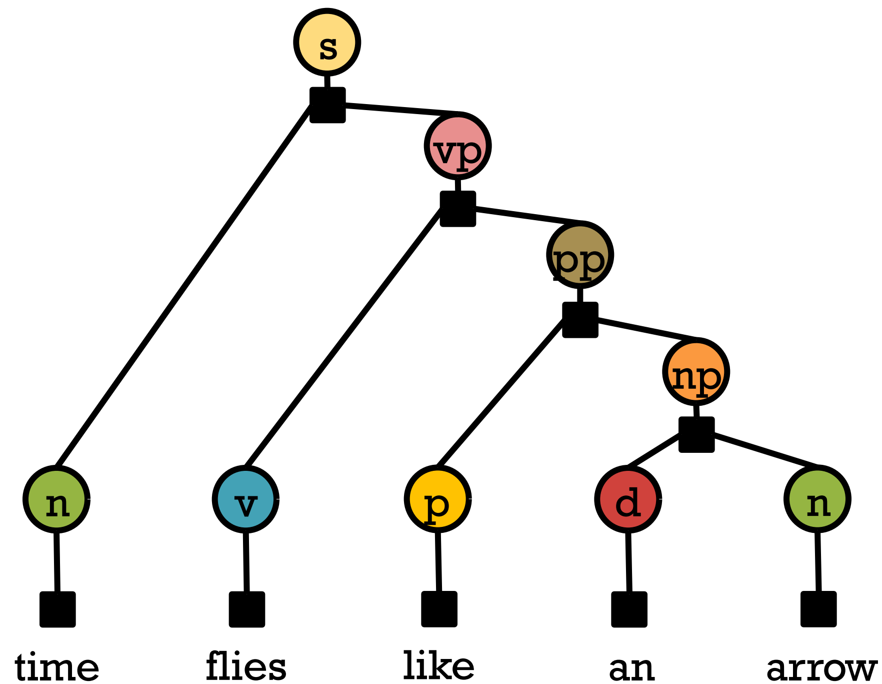
Construct a factor graph which mimics the tree structure, to **predict** the tags / nonterminals.



Annotating a Tree

Given: a **sentence** and unlabeled **parse** tree.

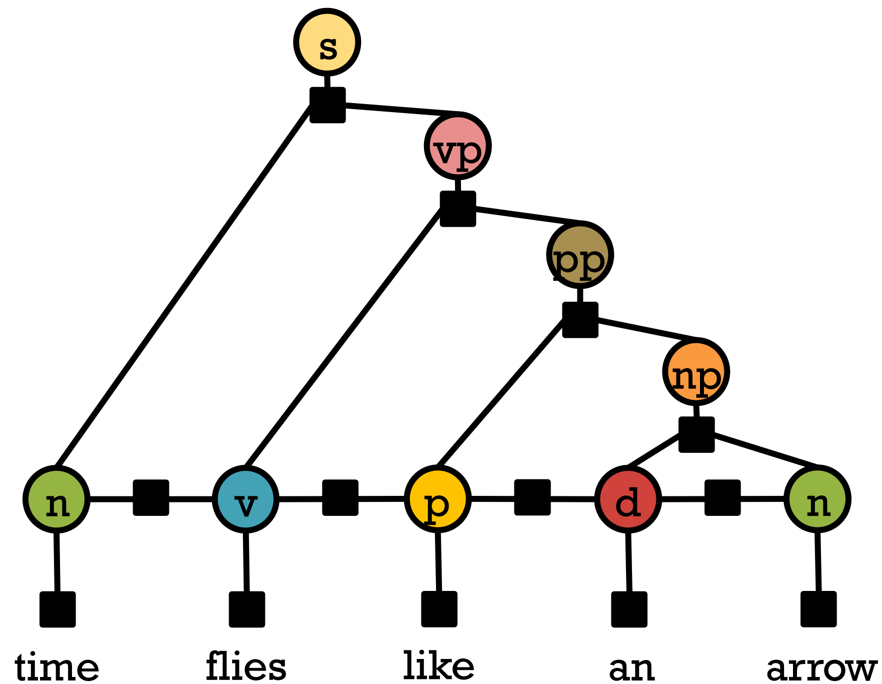
Construct a factor graph which mimics the tree structure, to **predict** the tags / nonterminals.



Annotating a Tree

Given: a **sentence** and unlabeled **parse** tree.

Construct a factor graph which mimics the tree structure, to **predict** the tags / nonterminals.



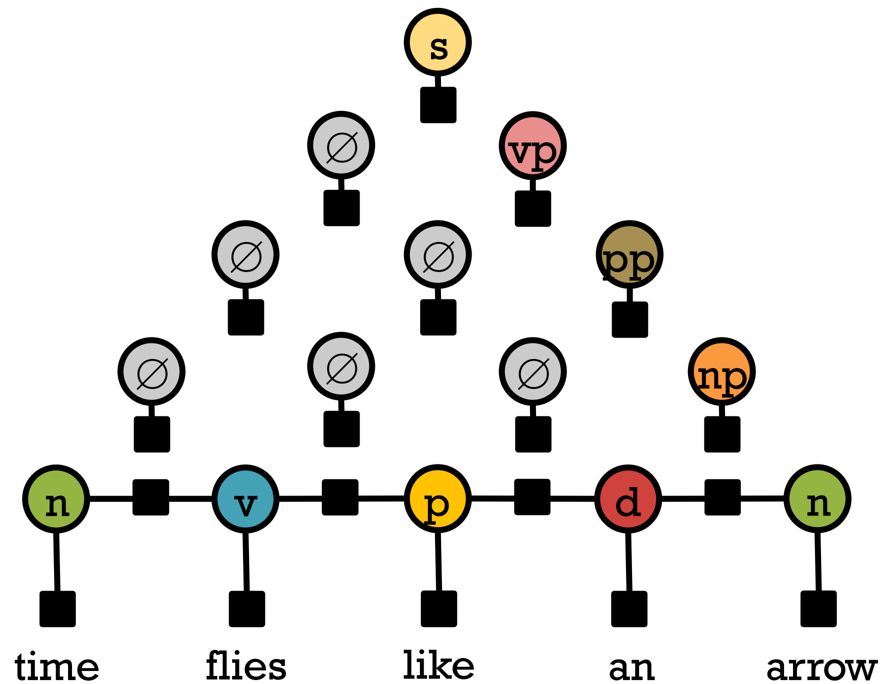
We could add a **linear chain** structure between tags.

(This creates **cycles**!)

Constituency Parsing

What if we needed to predict the tree structure too?

Use more variables:
Predict the nonterminal of each substring, or ∅ if it's not a constituent.

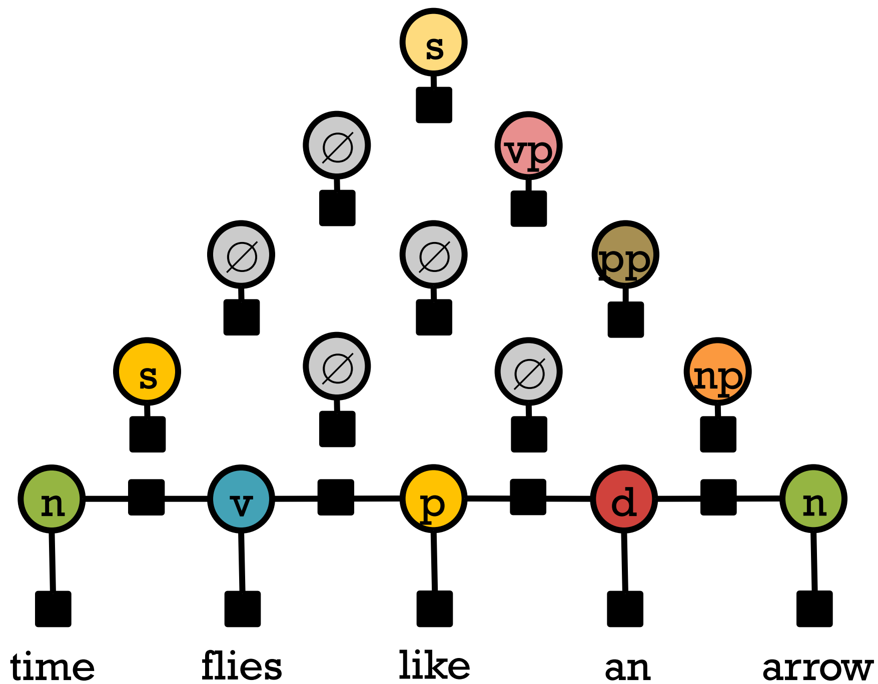


Constituency Parsing

What if we needed to predict the tree structure too?

Use more variables: Predict the nonterminal of each substring, or ∅ if it's not a constituent.

But nothing prevents non-tree structures.

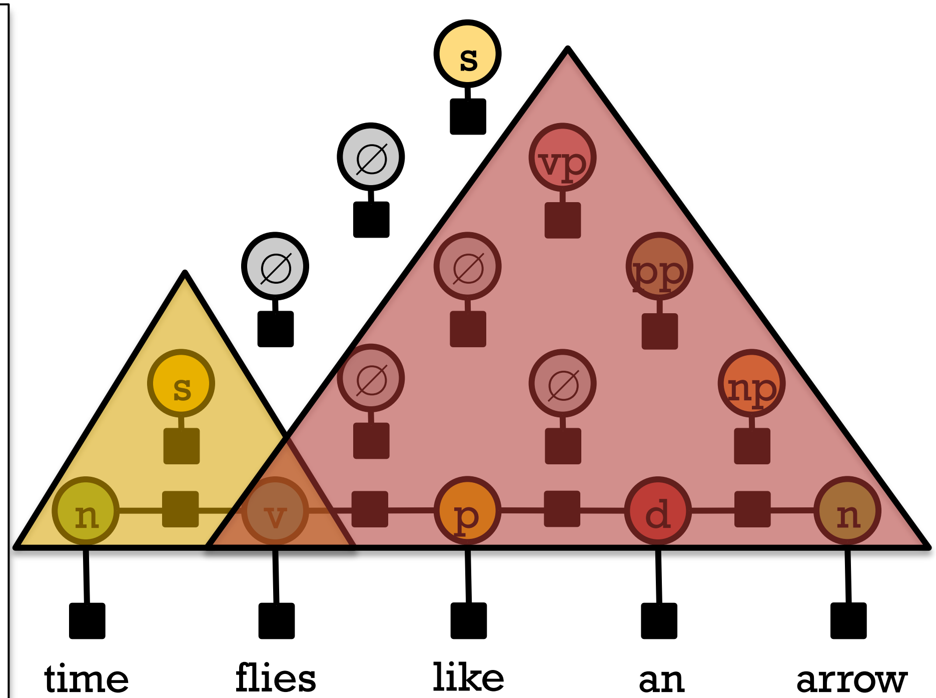


Constituency Parsing

What if we needed to predict the tree structure too?

Use more variables: Predict the nonterminal of each substring, or ∅ if it's not a constituent.

But nothing prevents non-tree structures.

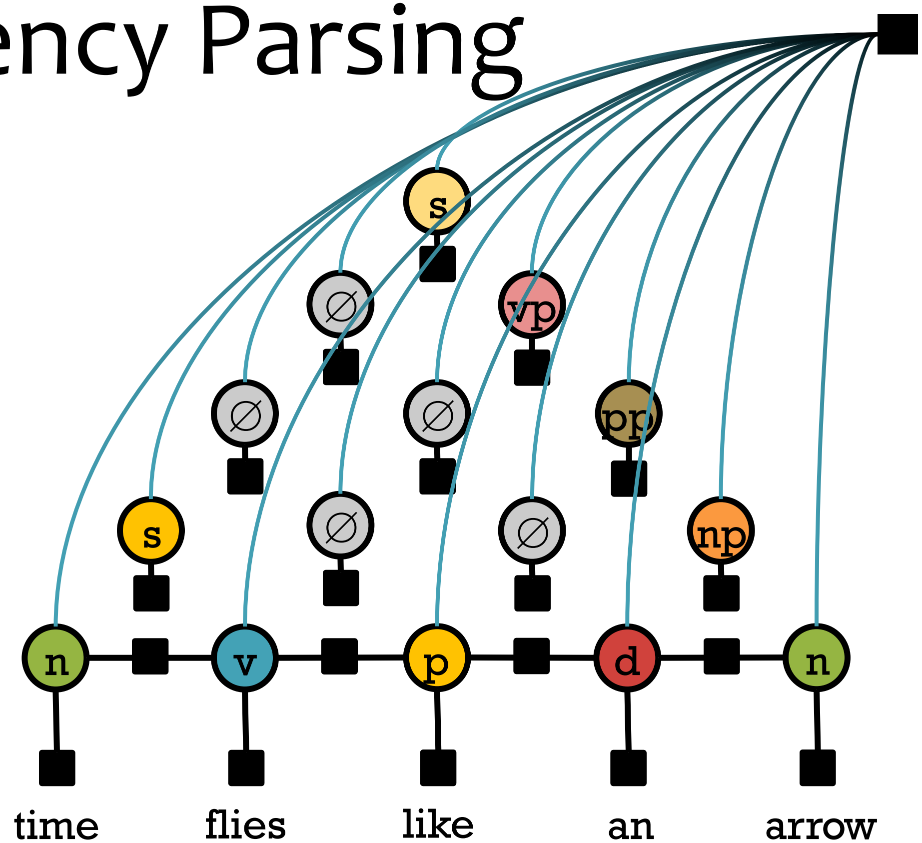


Constituency Parsing

What if we needed to predict the tree structure too?

Use more variables: Predict the nonterminal of each substring, or ∅ if it's not a constituent.

But nothing prevents non-tree structures.



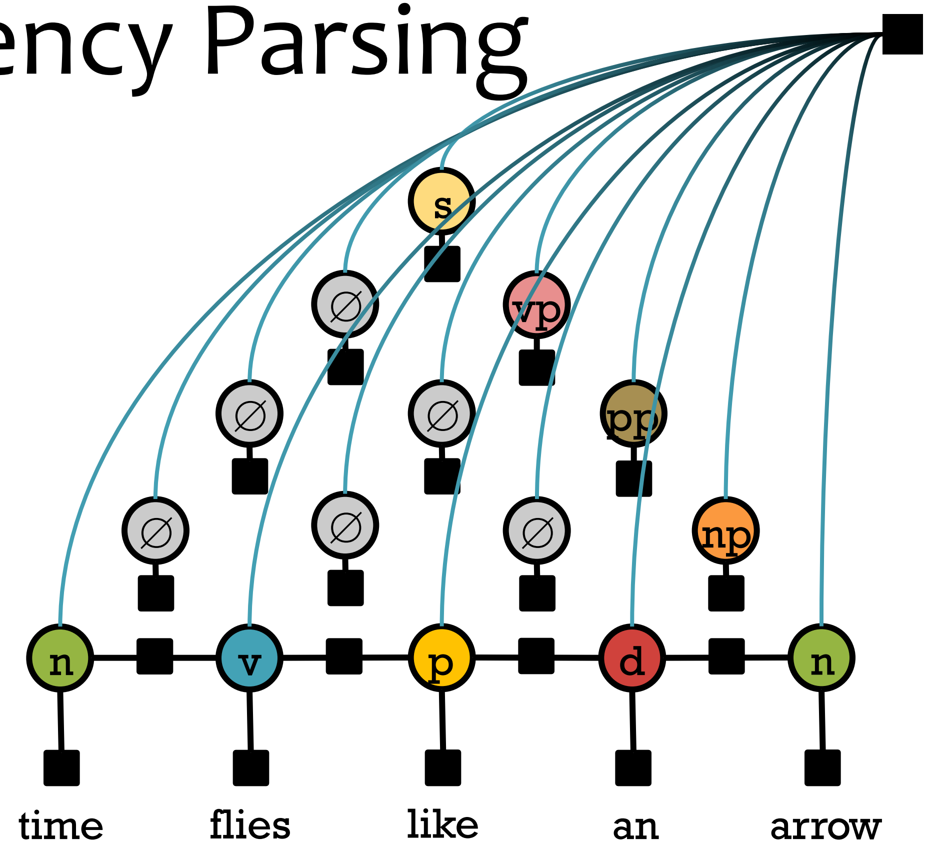
Add a factor which multiplies in *1* if the variables form a tree and *0* otherwise.

Constituency Parsing

What if we needed to predict the tree structure too?

Use more variables: Predict the nonterminal of each substring, or ∅ if it's not a constituent.

But nothing prevents non-tree structures.

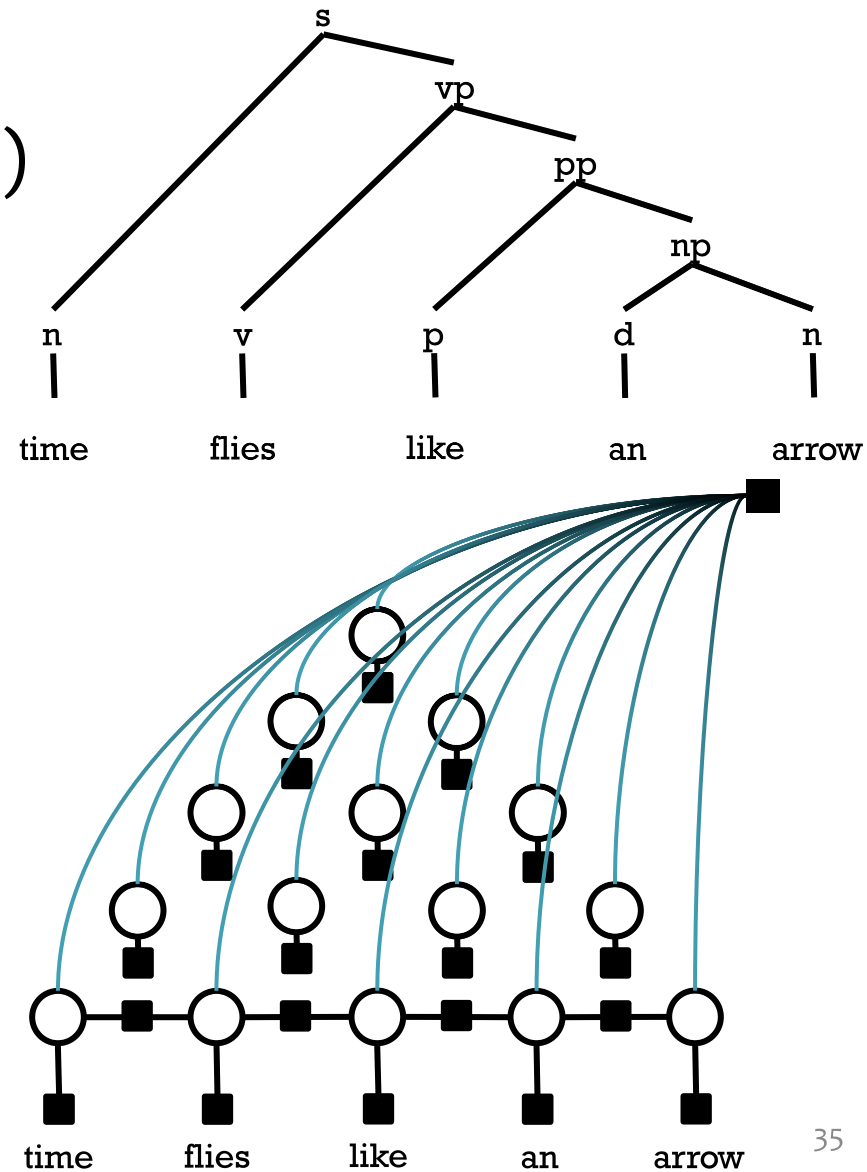


Add a factor which multiplies in *1* if the variables form a tree and *0* otherwise.

Constituency Parsing

- **Variables:**
  – Constituent type (or ∅) for each of $O(n^2)$ substrings
- **Interactions:**
  – Constituents must describe a binary tree
  – Tag bigrams
  – Nonterminal triples (parent, left-child, right-child)

[these factors not shown]

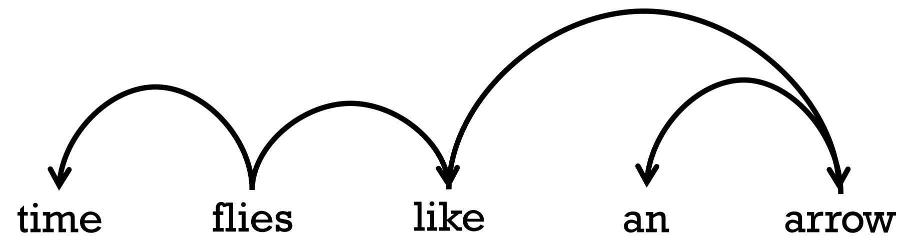


(Naradowsky, Vieira, & Smith, 2012)

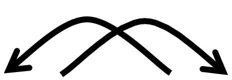
Example Task:

Dependency Parsing

- **Variables:**
  – POS tag for each word
  – Syntactic label (or ∅) for each of $O(n^2)$ possible directed arcs



- **Interactions:**
  – Arcs must form a tree
  – Discourage (or forbid) crossing edges



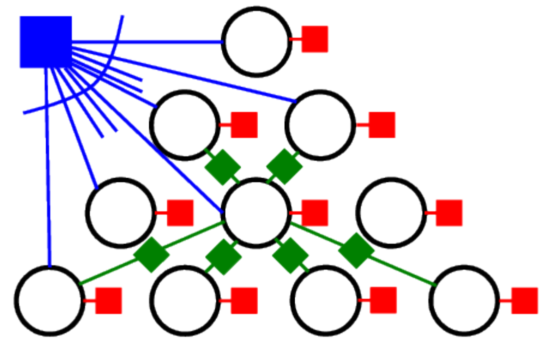
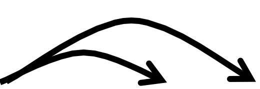
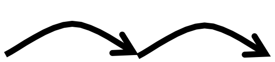


*Figure from Burkett & Klein (2012)

  – Features on edge pairs that share a vertex





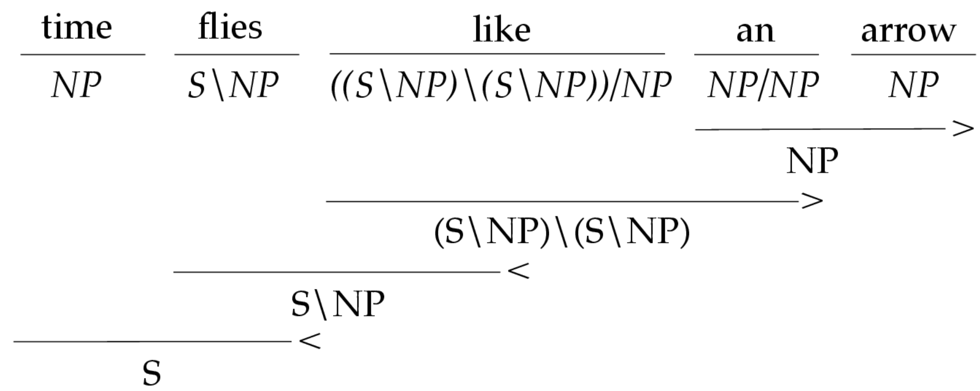
    - Learn to *discourage* a verb from having 2 objects, etc.
    - Learn to *encourage* specific multi-arc constructions

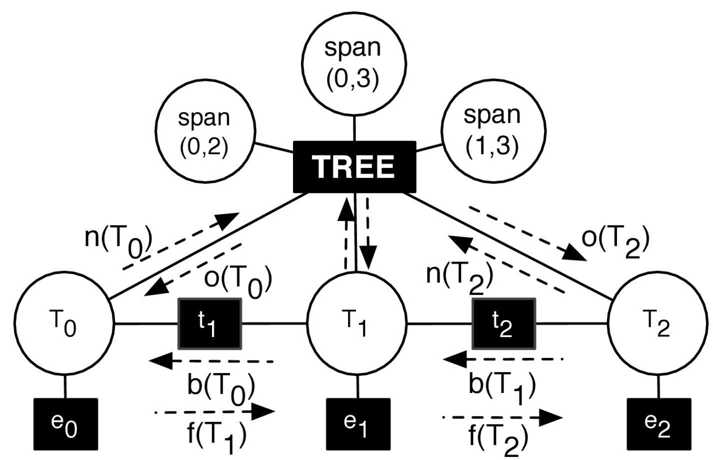
(Smith & Eisner, 2008)

Example Task:

Joint CCG Parsing and Supertagging

- **Variables:**
  - Spans
  - Labels on non-terminals
  - Supertags on pre-terminals
- **Interactions:**
  - Spans must form a tree
  - Triples of labels: parent, left-child, and right-child
  - Adjacent tags





(Auli & Lopez, 2011)

Example task:

Transliteration or Back-Transliteration

- **Variables (string):**
  - English and Japanese orthographic strings
  - English and Japanese phonological strings
- **Interactions:**
  - All pairs of strings could be relevant

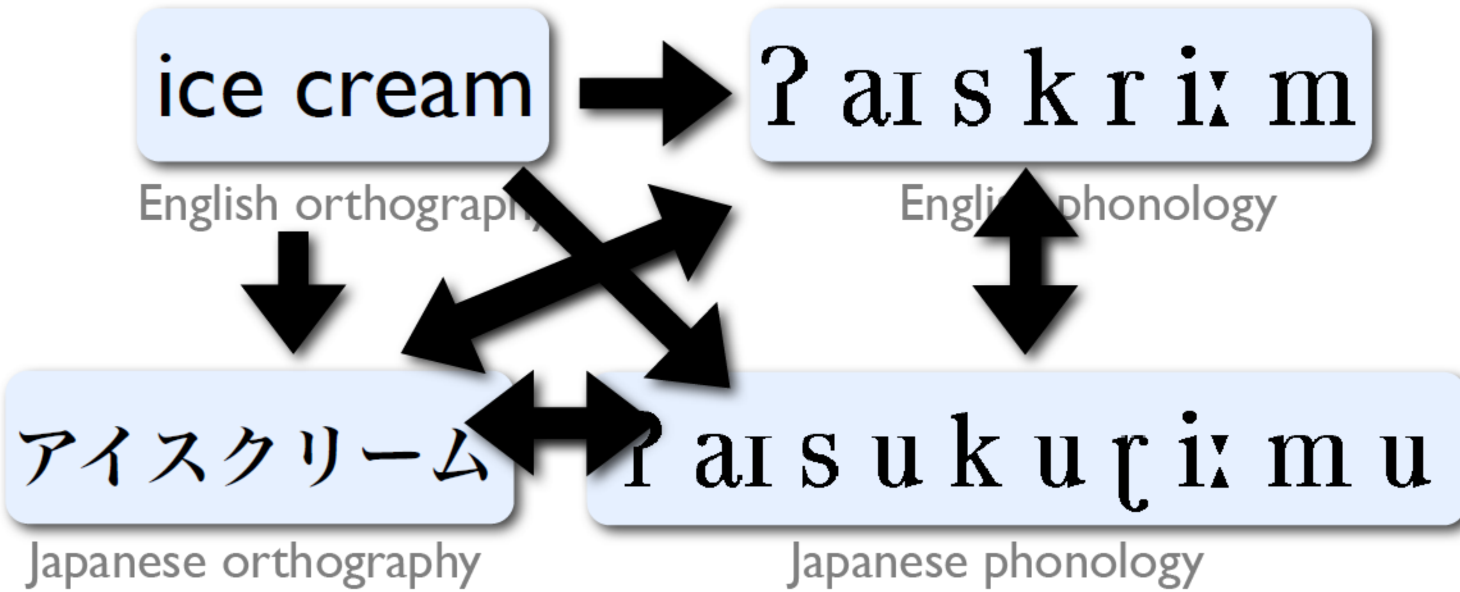
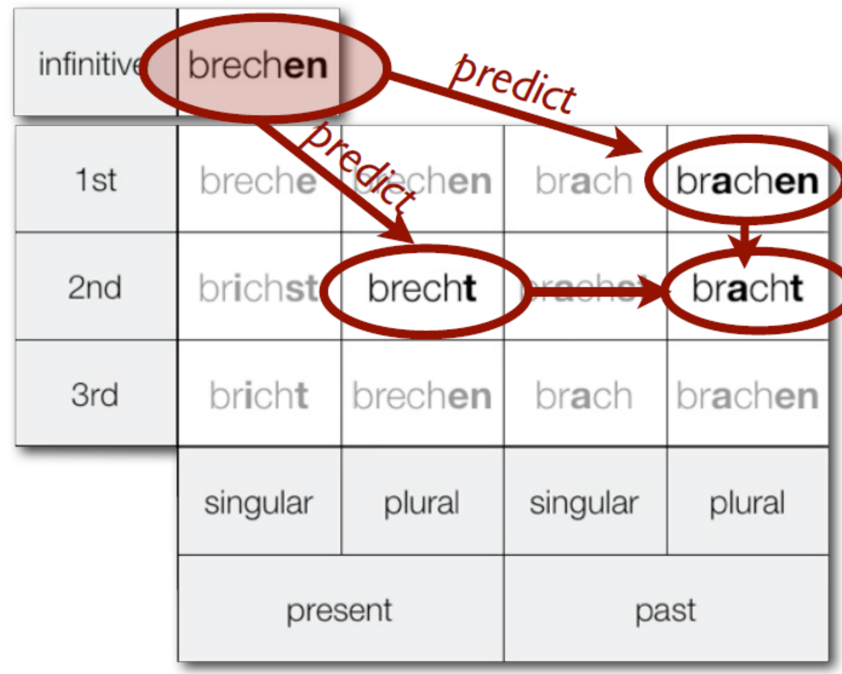


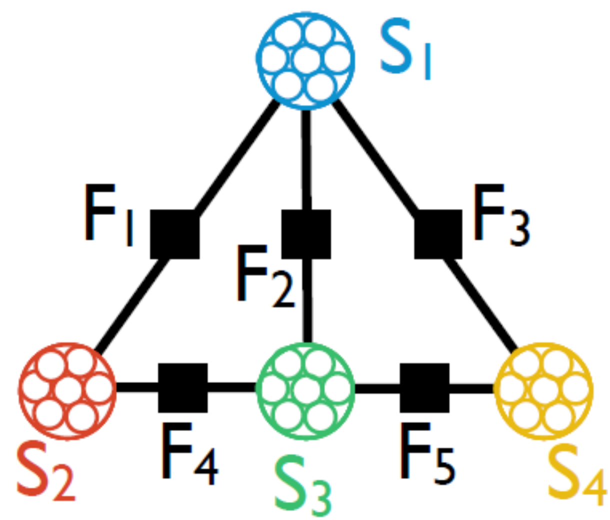
Figure thanks to Markus Dreyer

Example task:

Morphological Paradigms

- **Variables (string):**
  - Inflected forms of the same verb
- **Interactions:**
  - Between pairs of entries in the table (e.g. infinitive form affects present-singular)

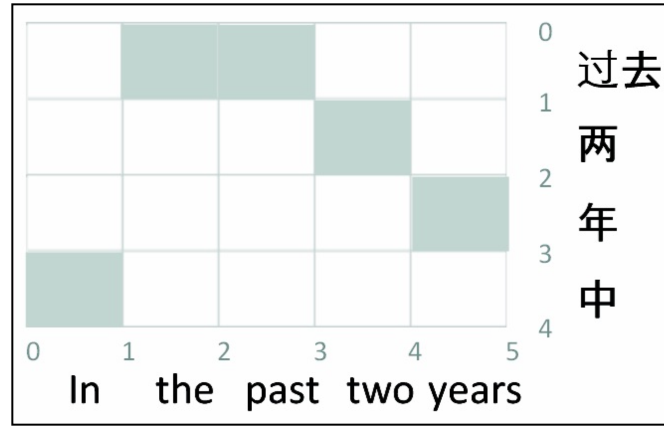


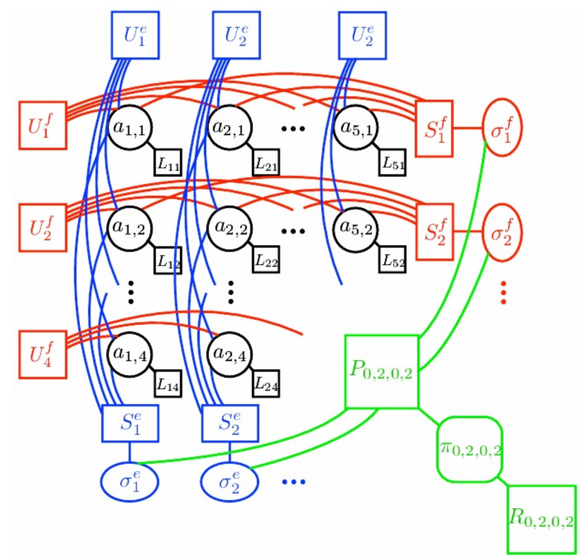


(Dreyer & Eisner, 2009)

Word Alignment / Phrase Extraction

- **Variables (boolean):**
  - For each (Chinese phrase, English phrase) pair, are they linked?
- **Interactions:**
  - Word fertilities
  - Few "jumps" (discontinuities)
  - Syntactic reorderings
  - "ITG contraint" on alignment
  - Phrases are disjoint (?)

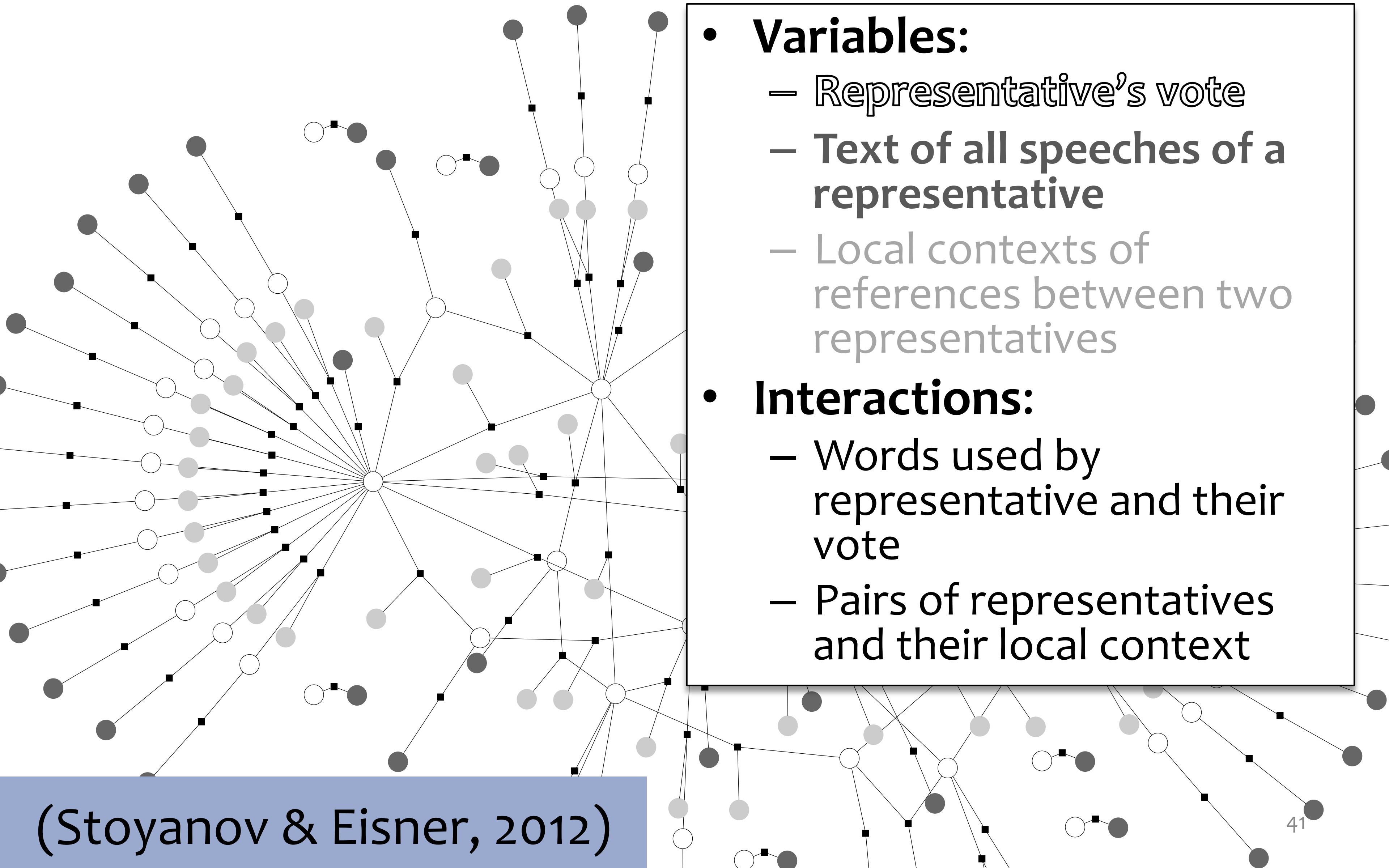




(Burkett & Klein, 2012)

Application:

Congressional Voting



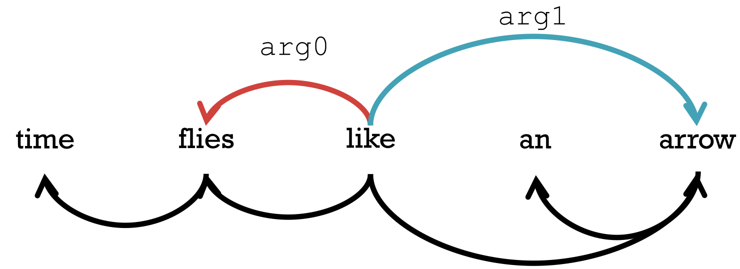
- **Variables:**
  - Representative's vote
  - **Text of all speeches of a representative**
  - Local contexts of references between two representatives
- **Interactions:**
  - Words used by representative and their vote
  - Pairs of representatives and their local context

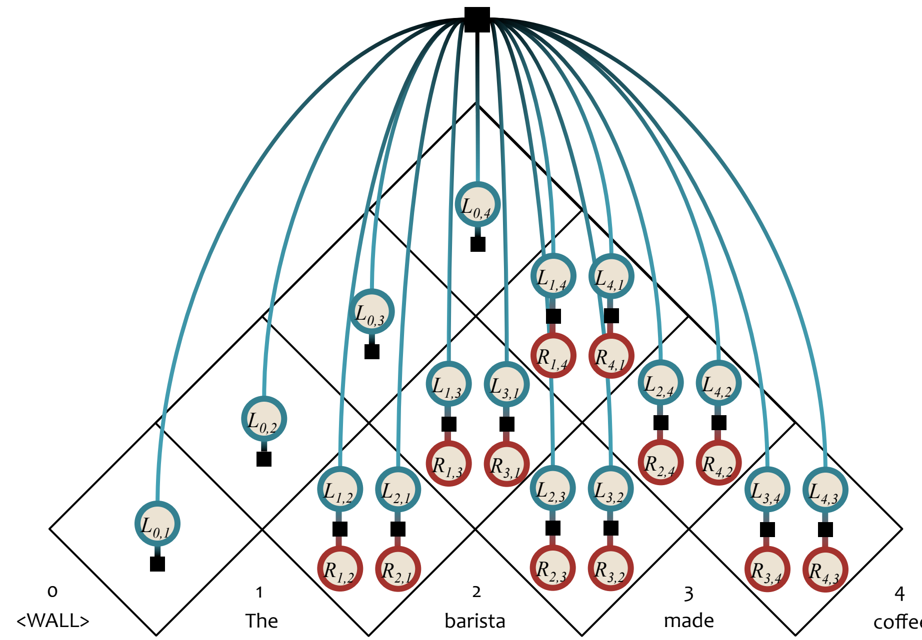
(Stoyanov & Eisner, 2012)

Application:

Semantic Role Labeling with Latent Syntax

- **Variables:**
  - Semantic predicate sense
  - Semantic dependency arcs
  - Labels of semantic arcs
  - Latent syntactic dependency arcs
- **Interactions:**
  - Pairs of syntactic and semantic dependencies
  - Syntactic dependency arcs must form a tree

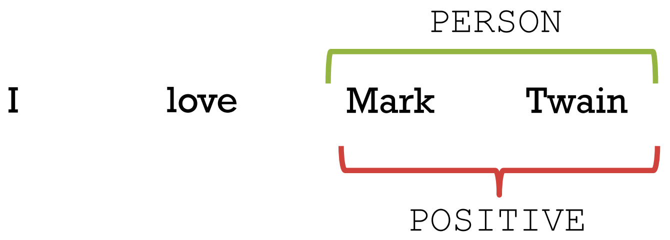


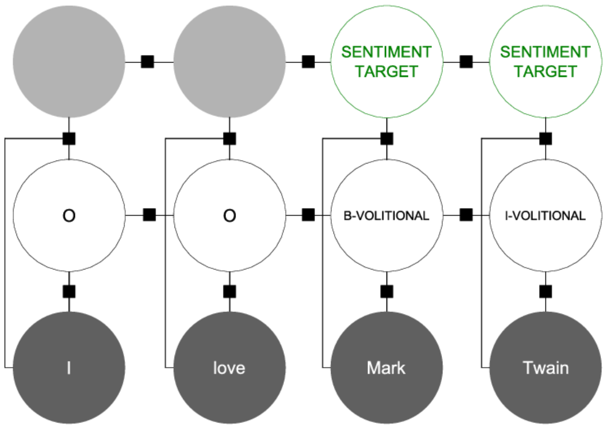


(Naradowsky, Riedel, & Smith, 2012)
(Gormley, Mitchell, Van Durme, & Dredze, 2014)

Joint NER & Sentiment Analysis

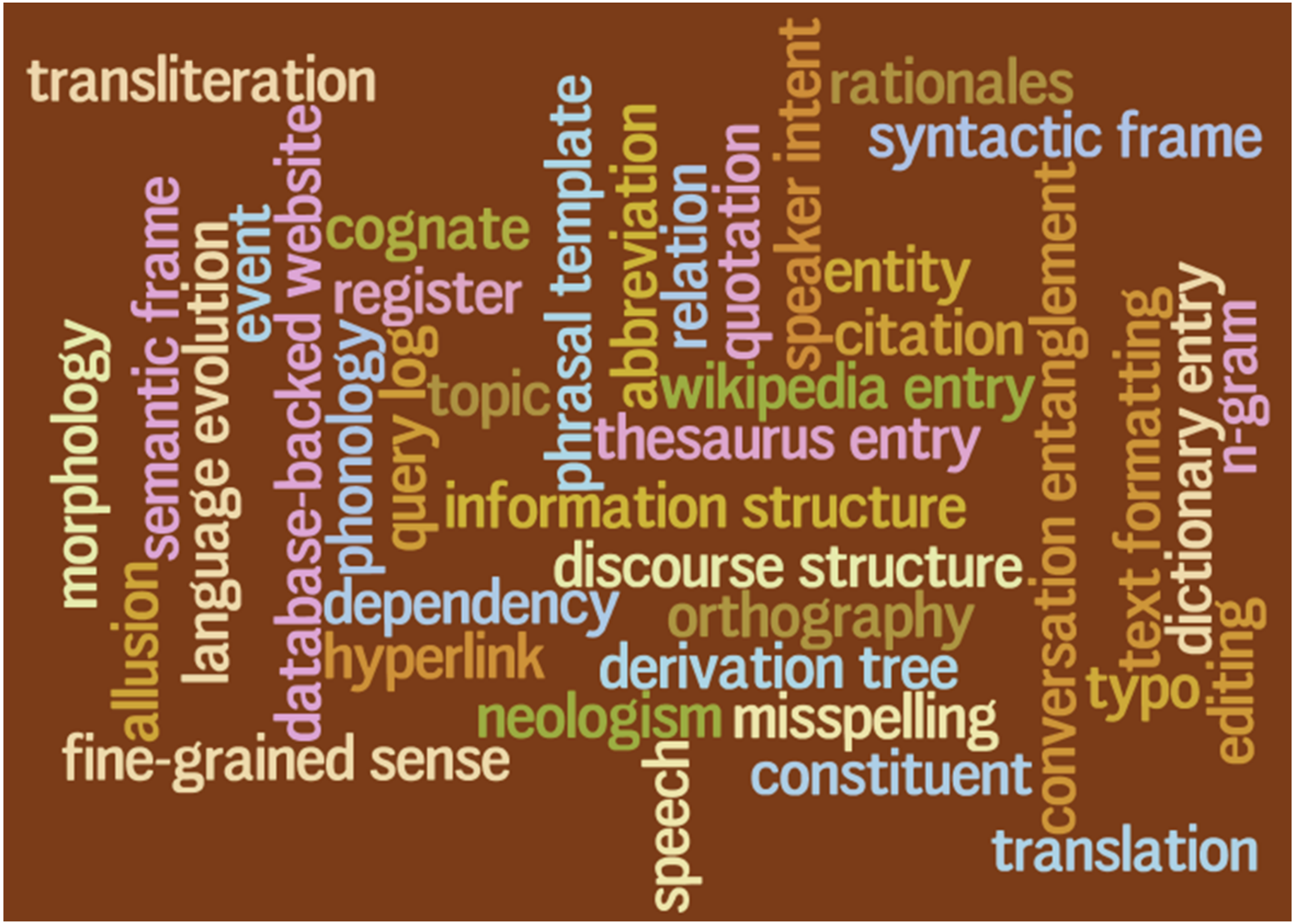
- **Variables:**
  - Named entity spans
  - Sentiment directed toward each entity
- **Interactions:**
  - Words and entities
  - Entities and sentiment



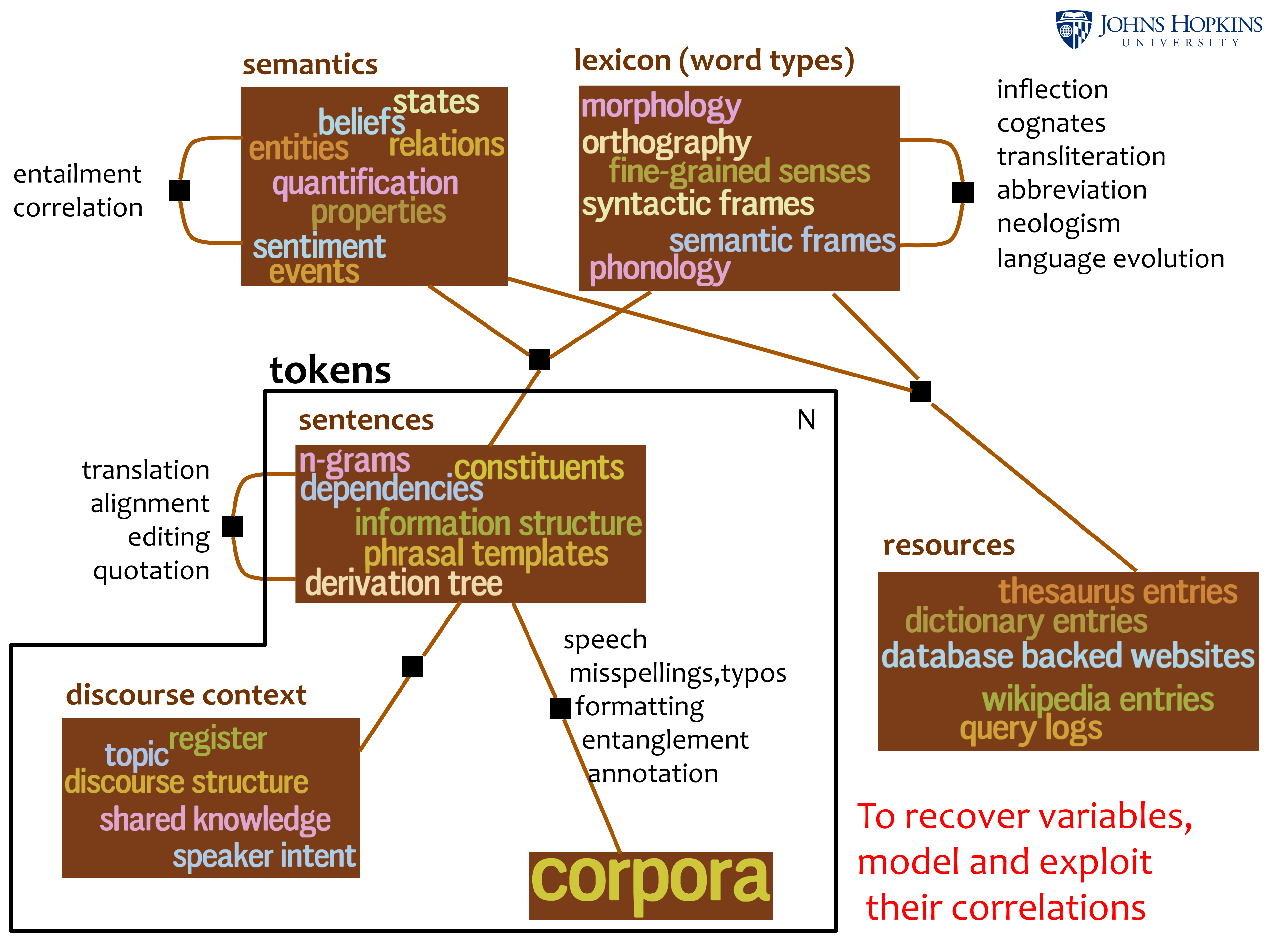


(Mitchell, Aguilar, Wilson, & Van Durme, 2013)

Variable-centric view of the world



When we deeply understand language, what representations (type and token) does that understanding comprise?


JOHNS HOPKINS UNIVERSITY
semantics
states
beliefs
entities
relations
quantification
properties
sentiment
events
entailment
correlation
lexicon (word types)
morphology
orthography
fine-grained senses
syntactic frames
semantic frames
phonology
inflection
cognates
transliteration
abbreviation
neologism
language evolution
tokens
N
sentences
n-grams
constituents
dependencies
information structure
phrasal templates
derivation tree
translation
alignment
editing
quotation
resources
thesaurus entries
dictionary entries
database backed websites
wikipedia entries
query logs
speech
misspellings,typos
formatting
entanglement
annotation
discourse context
register
topic
discourse structure
shared knowledge
speaker intent
corpora
To recover variables, model and exploit their correlations

Section 2:
Belief Propagation Basics

Outline

- Do you want to push past the simple NLP models (logistic regression, PCFG, etc.) that we've all been using for 20 years?
- Then this tutorial is extremely practical for you!
  1. **Models:** Factor graphs can express interactions among linguistic structures.
  2. **Algorithm:** BP estimates the global effect of these interactions on each variable, using local computations.
  3. **Intuitions:** What's going on here? Can we trust BP's estimates?
  4. **Fancier Models:** Hide a whole grammar and dynamic programming algorithm within a single factor. BP coordinates multiple factors.
  5. **Tweaked Algorithm:** Finish in fewer steps and make the steps faster.
  6. **Learning:** Tune the parameters. Approximately improve the true predictions -- or truly improve the approximate predictions.
  7. **Software:** Build the model you want!

Outline

- Do you want to push past the simple NLP models (logistic regression, PCFG, etc.) that we've all been using for 20 years?
- Then this tutorial is extremely practical for you!
  1. **Models:** Factor graphs can express interactions among linguistic structures.
  2. **Algorithm:** BP estimates the global effect of these interactions on each variable, using local computations.
  3. **Intuitions:** What's going on here? Can we trust BP's estimates?
  4. **Fancier Models:** Hide a whole grammar and dynamic programming algorithm within a single factor. BP coordinates multiple factors.
  5. **Tweaked Algorithm:** Finish in fewer steps and make the steps faster.
  6. **Learning:** Tune the parameters. Approximately improve the true predictions -- or truly improve the approximate predictions.
  7. **Software:** Build the model you want!

Factor Graph Notation

- Variables:

$$\mathcal{X} = \{X_1, \ldots, X_i, \ldots, X_n\}$$

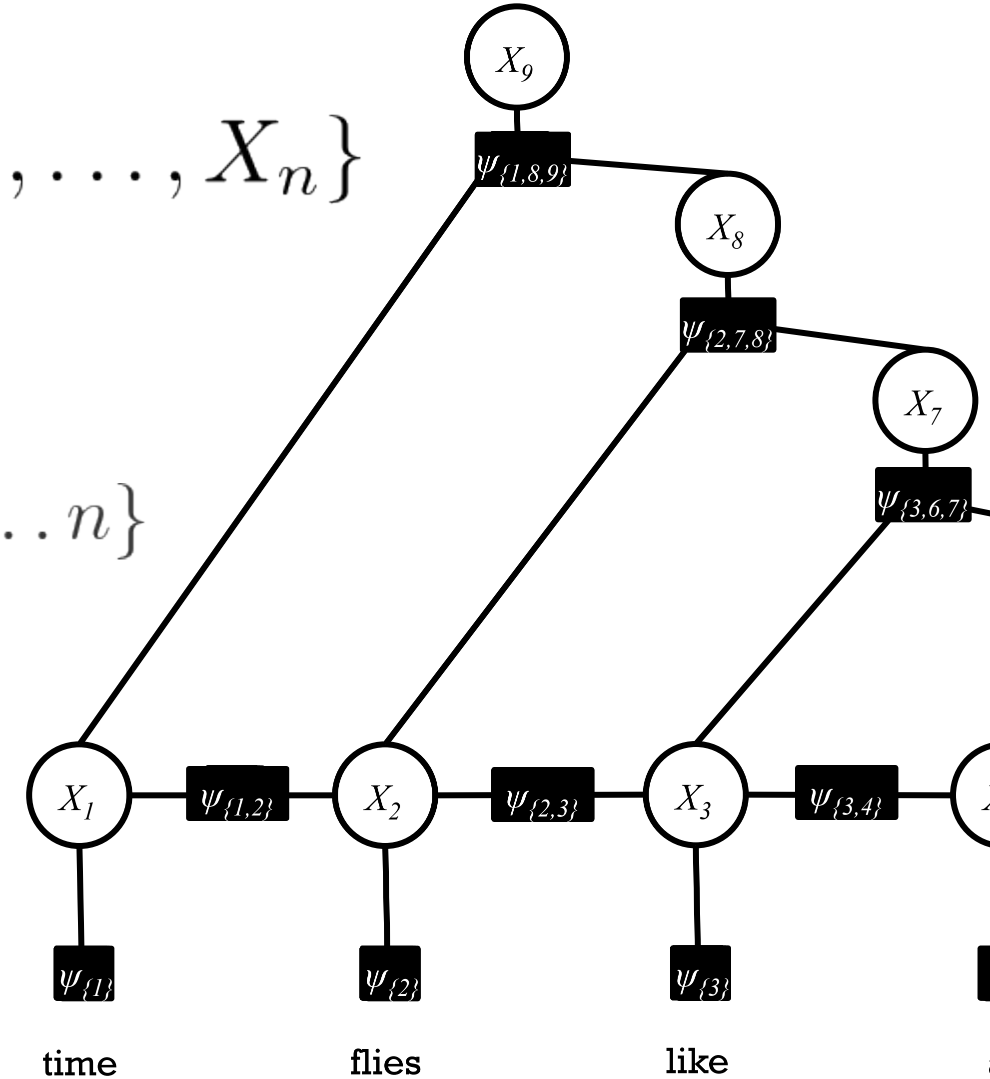
- Factors:

$$\psi_\alpha, \psi_\beta, \psi_\gamma, \ldots$$

where $\alpha, \beta, \gamma, \ldots \subseteq \{1, \ldots n\}$

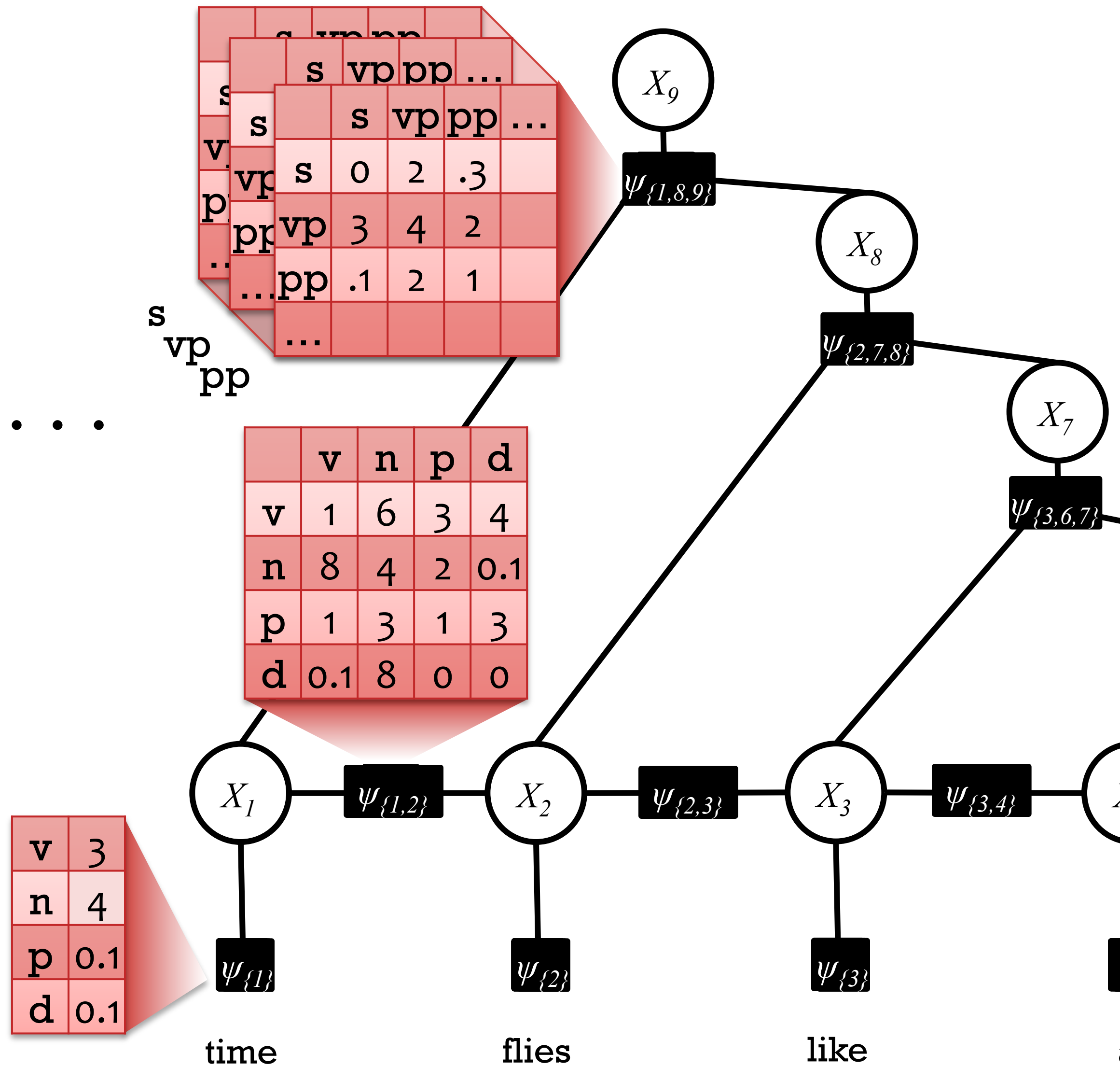
Joint Distribution

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



Factors are Tensors

- Factors:
  $\psi_\alpha, \psi_\beta, \psi_\gamma, \ldots$



Inference

Given a factor graph, two common tasks …

- Compute the most likely joint assignment,
  $\boldsymbol{x}^* = \text{argmax}_{\boldsymbol{x}}\, p(\boldsymbol{X}=\boldsymbol{x})$
- Compute the marginal distribution of variable X_i:
  $p(X_i=x_i)$ for each value x_i

Both consider *all* joint assignments.

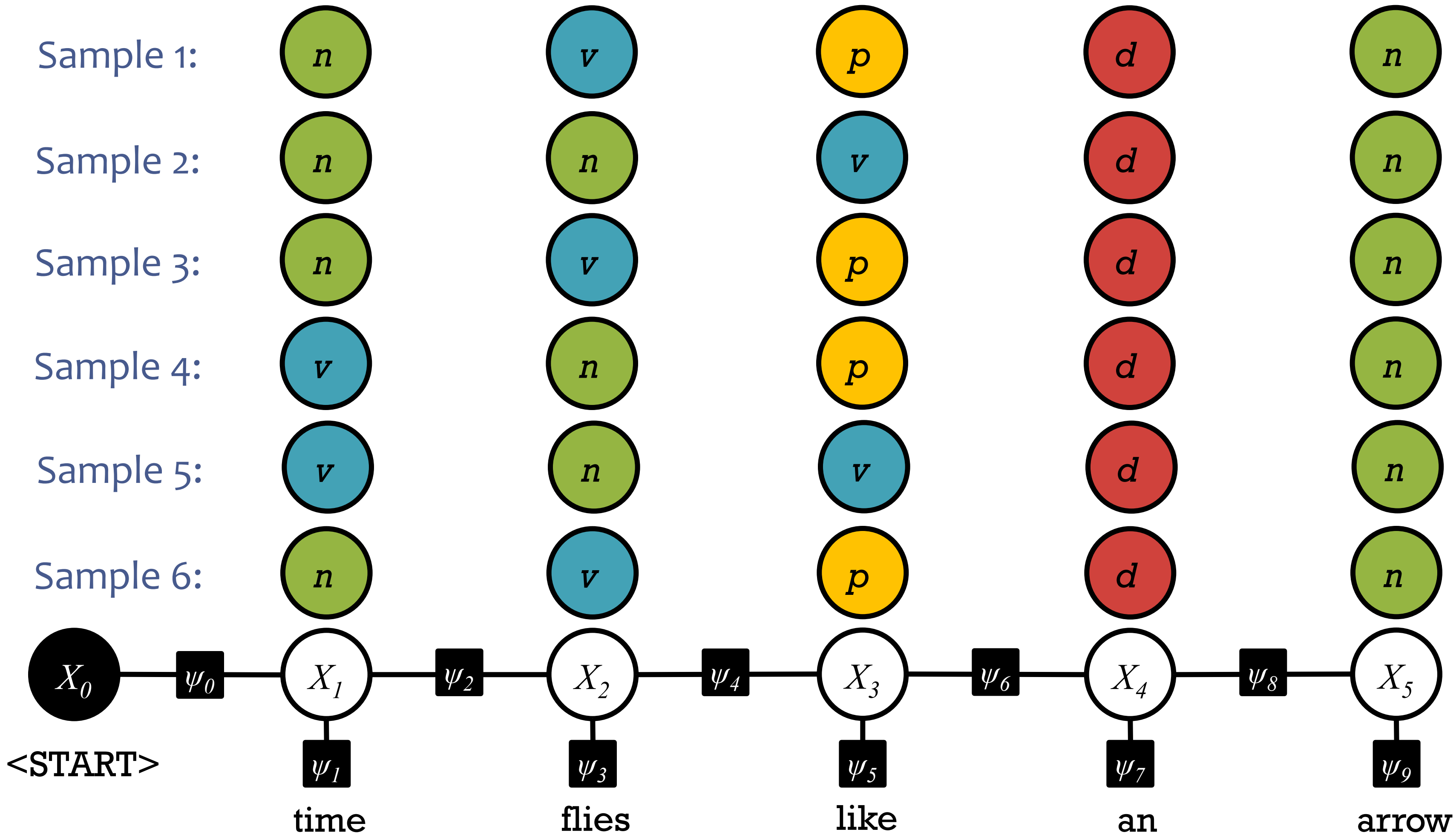
Both are NP-Hard in general.

So, we turn to **approximations**.

$p(X_i=x_i)$ = sum of $p(\boldsymbol{X}=\boldsymbol{x})$ over joint assignments with $X_i=x_i$

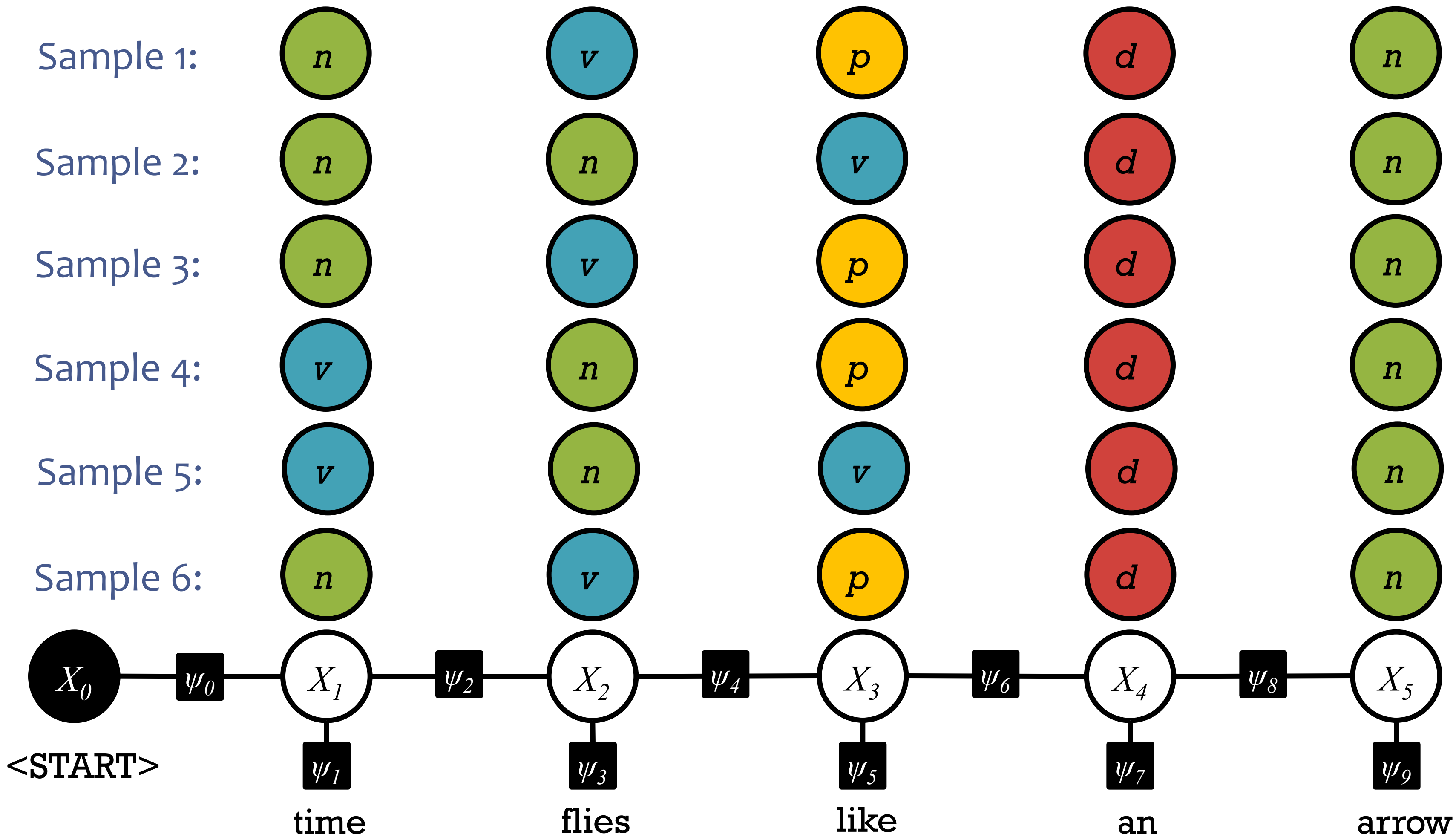
Marginals by Sampling on Factor Graph

Suppose we took many samples from the distribution over taggings: $p(\boldsymbol{x}) = \frac{1}{Z} \prod_{\alpha} \psi_\alpha(\boldsymbol{x_\alpha})$

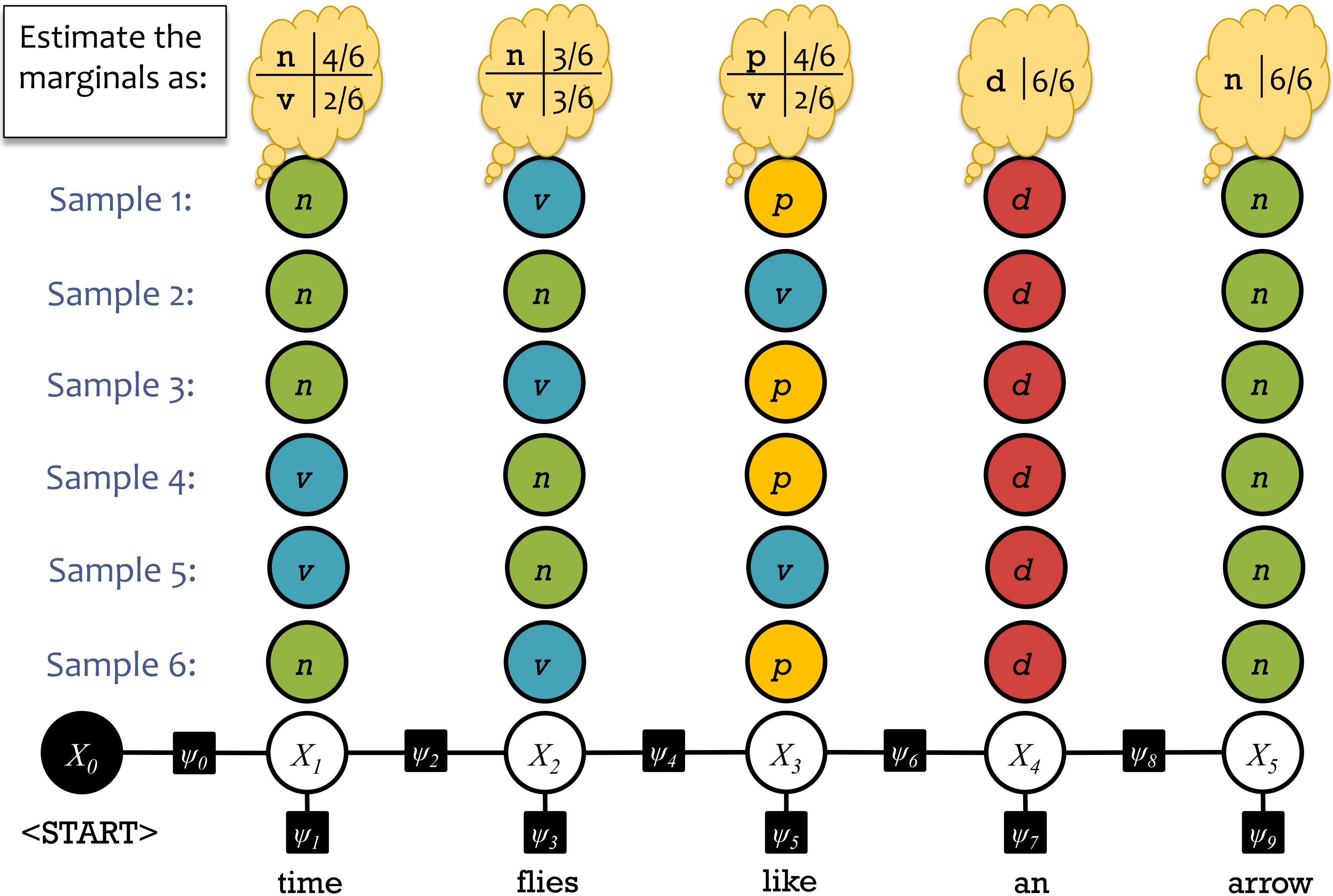


Marginals by Sampling on Factor Graph

The marginal $p(X_i = x_i)$ gives the probability that variable X_i takes value x_i in a random sample



Marginals by Sampling on Factor Graph



How do we get marginals without sampling?

That's what Belief Propagation is all about!

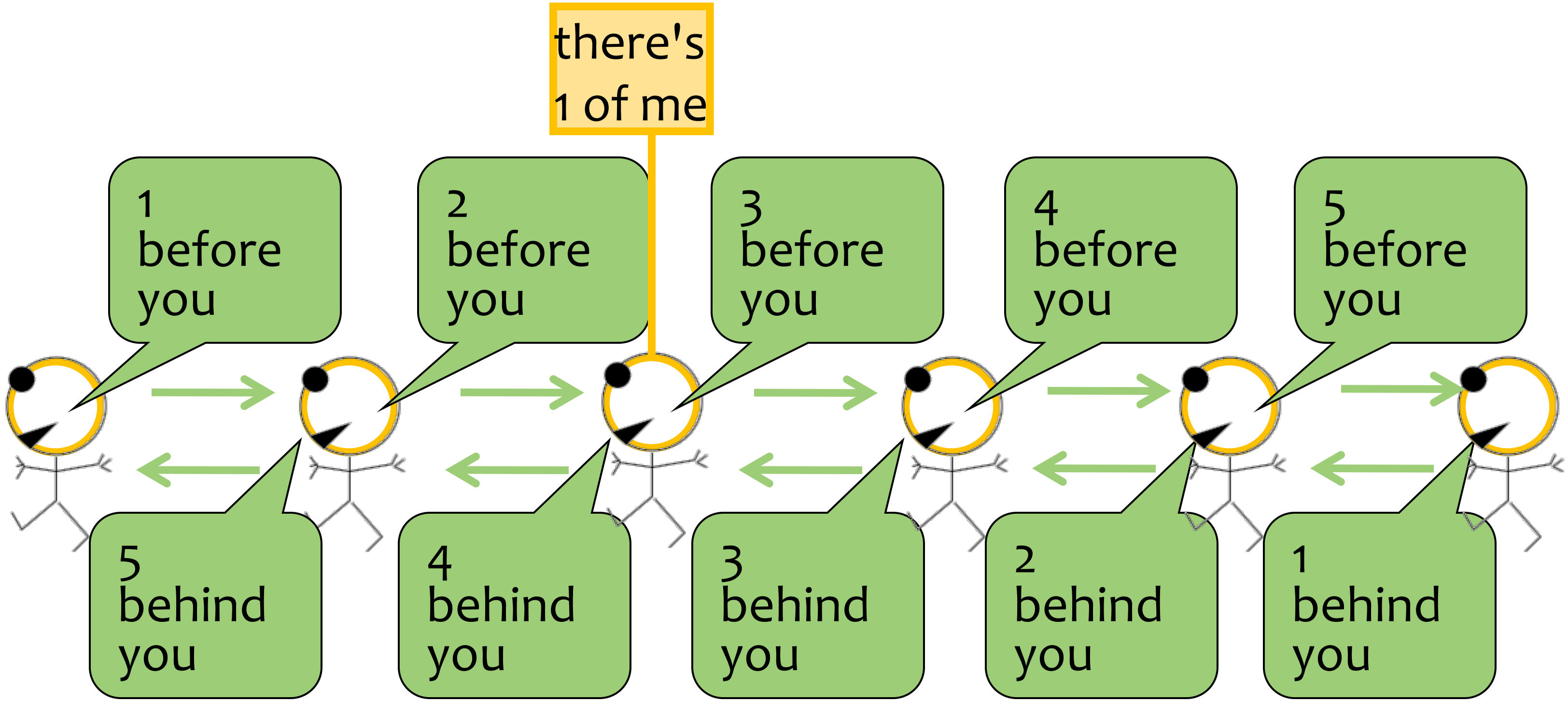
Why not just sample?

- Sampling one joint assignment is *also* NP-hard in general.
  - In practice: Use MCMC (e.g., Gibbs sampling) as an anytime algorithm.
  - So draw an approximate sample fast, or run longer for a "good" sample.

- Sampling finds the high-probability values x_i efficiently.
  But it takes too many samples to see the low-probability ones.
  - How do you find p("The quick brown fox …") under a language model?
    - Draw random sentences to see how often you get it? Takes a *long* time.
    - Or multiply factors (trigram probabilities)? That's what BP would do.

Great Ideas in ML: Message Passing

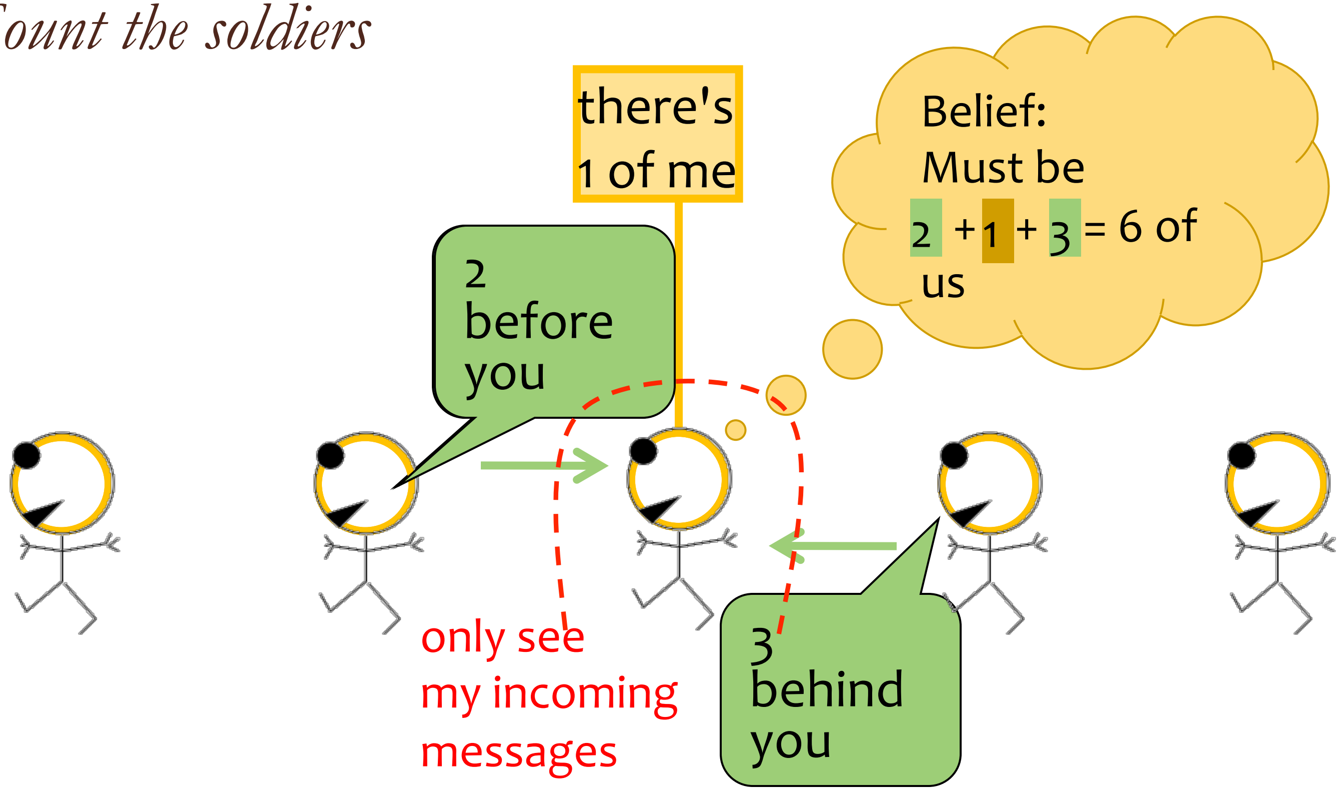
Count the soldiers



adapted from MacKay (2003) textbook

Great Ideas in ML: Message Passing

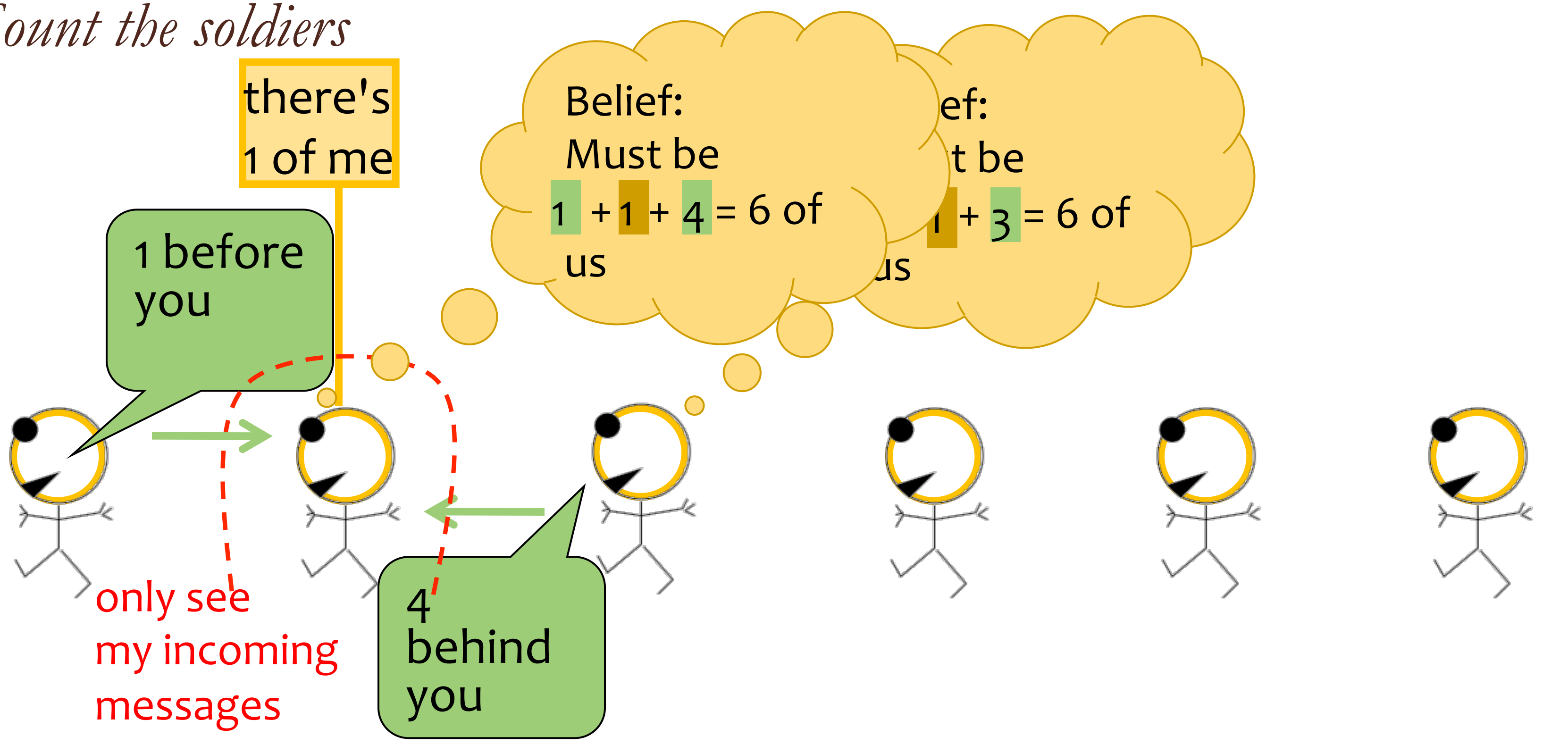
Count the soldiers



adapted from MacKay (2003) textbook

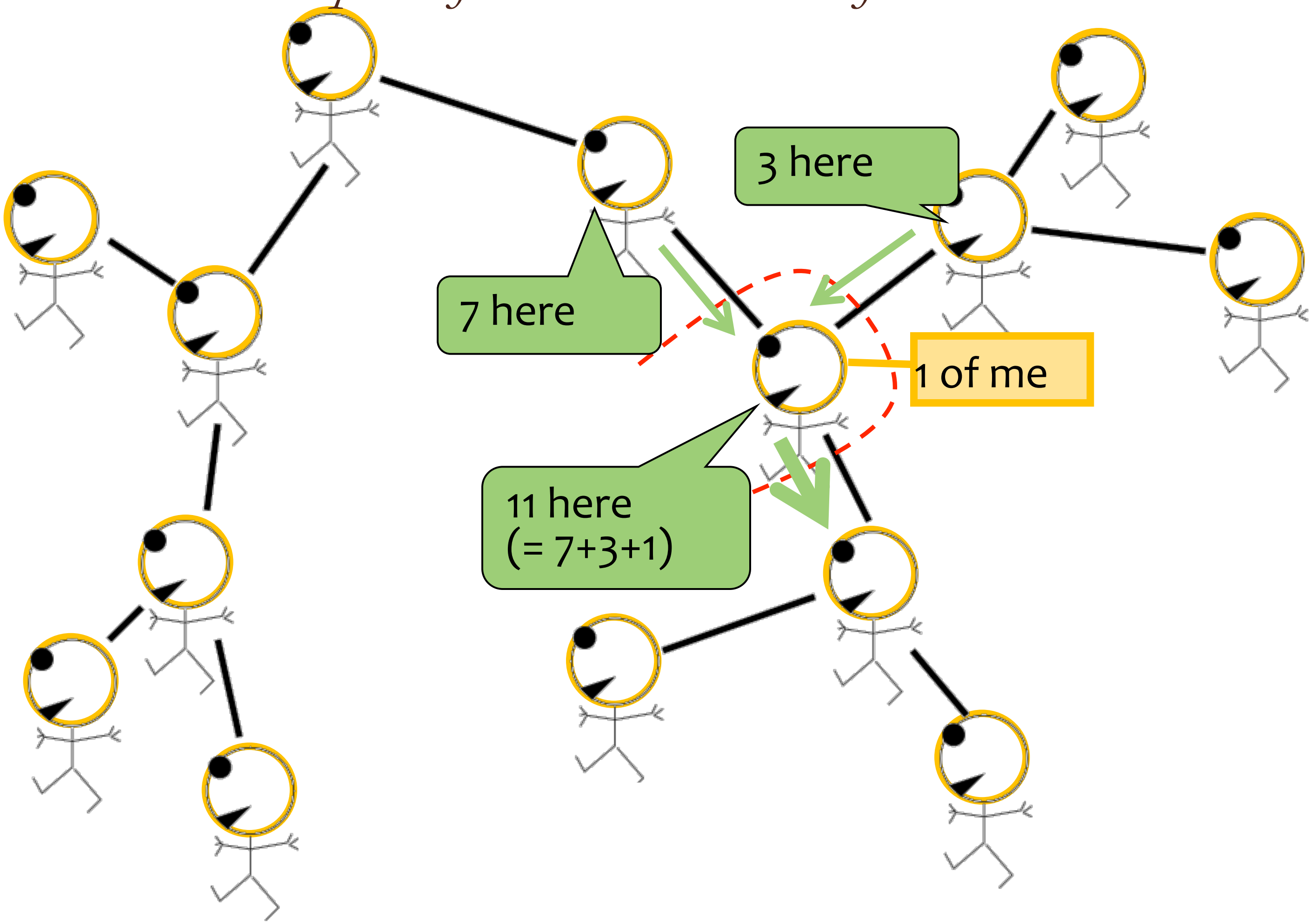
Great Ideas in ML: Message Passing

Count the soldiers



Great Ideas in ML: Message Passing

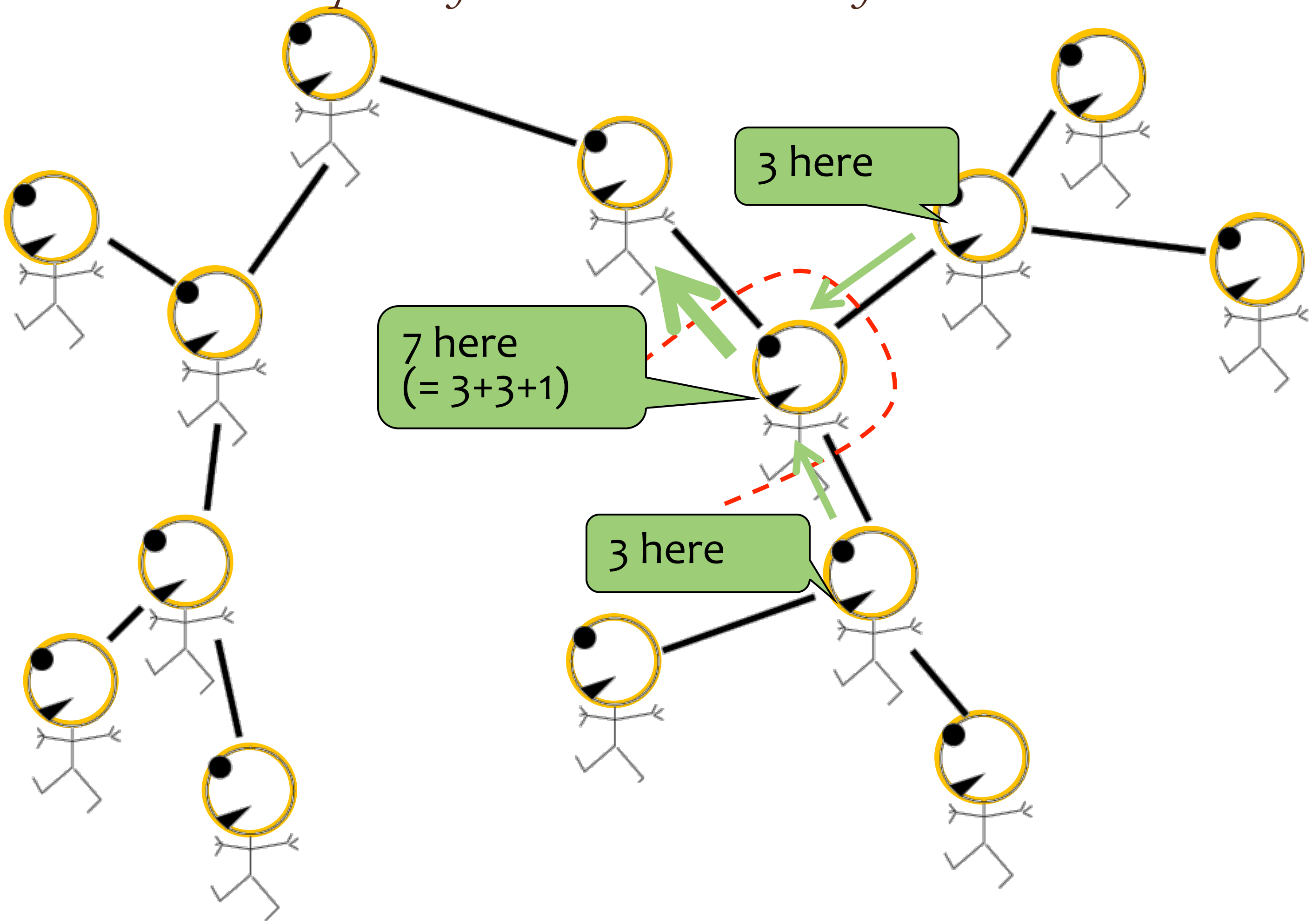
Each soldier receives reports from all branches of tree



adapted from MacKay (2003) textbook

Great Ideas in ML: Message Passing

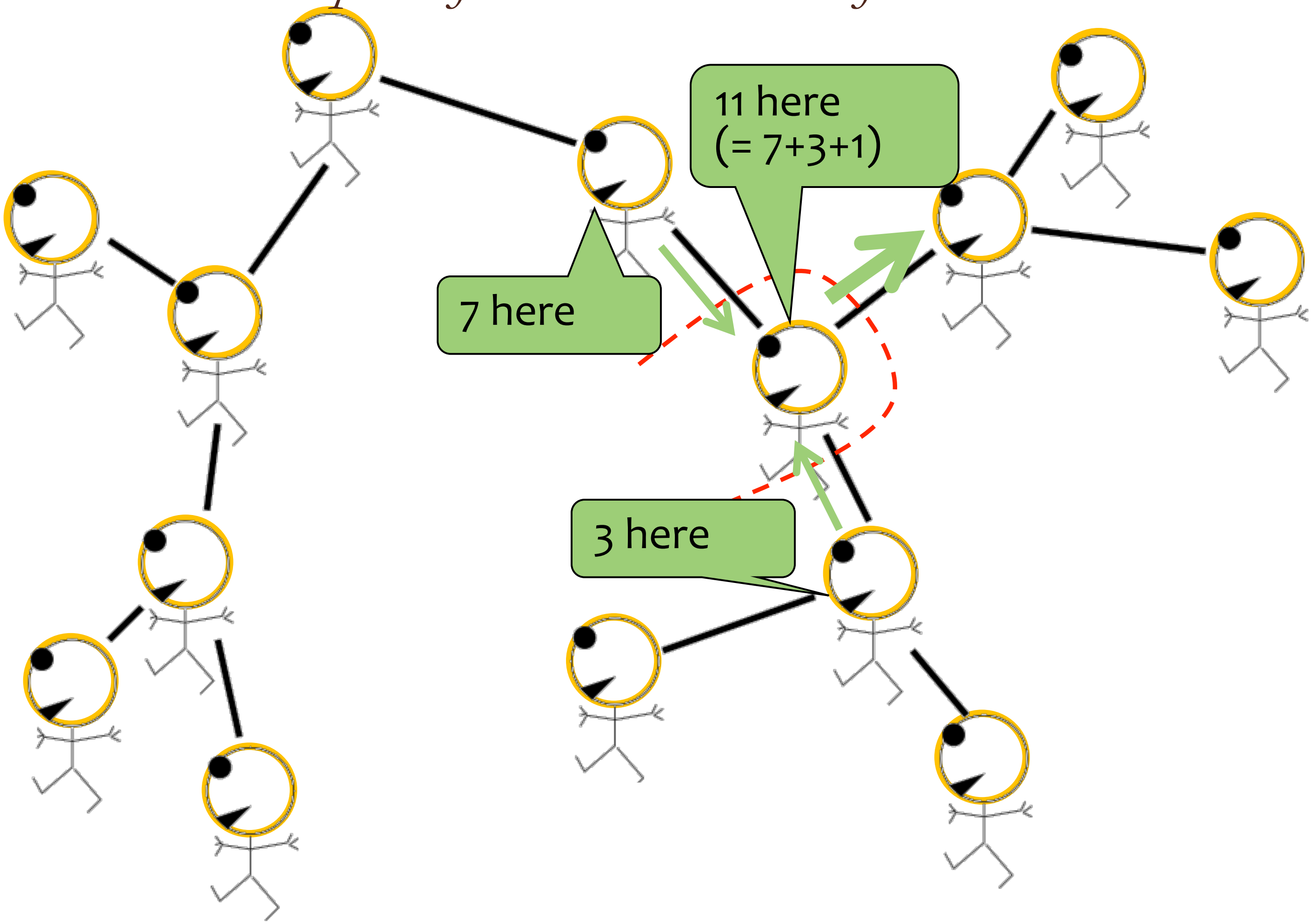
Each soldier receives reports from all branches of tree



adapted from MacKay (2003) textbook

Great Ideas in ML: Message Passing

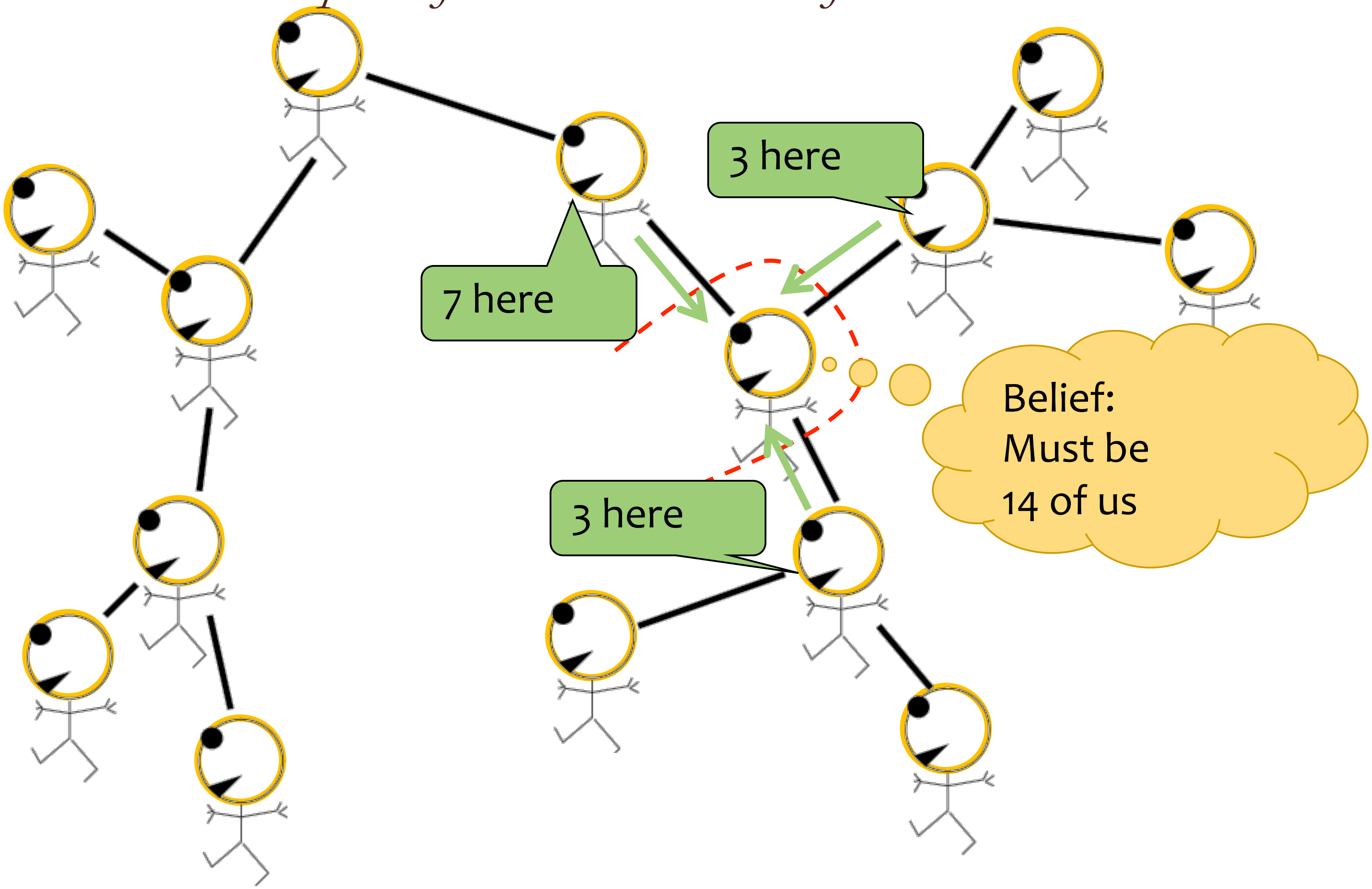
Each soldier receives reports from all branches of tree



adapted from MacKay (2003) textbook

Great Ideas in ML: Message Passing

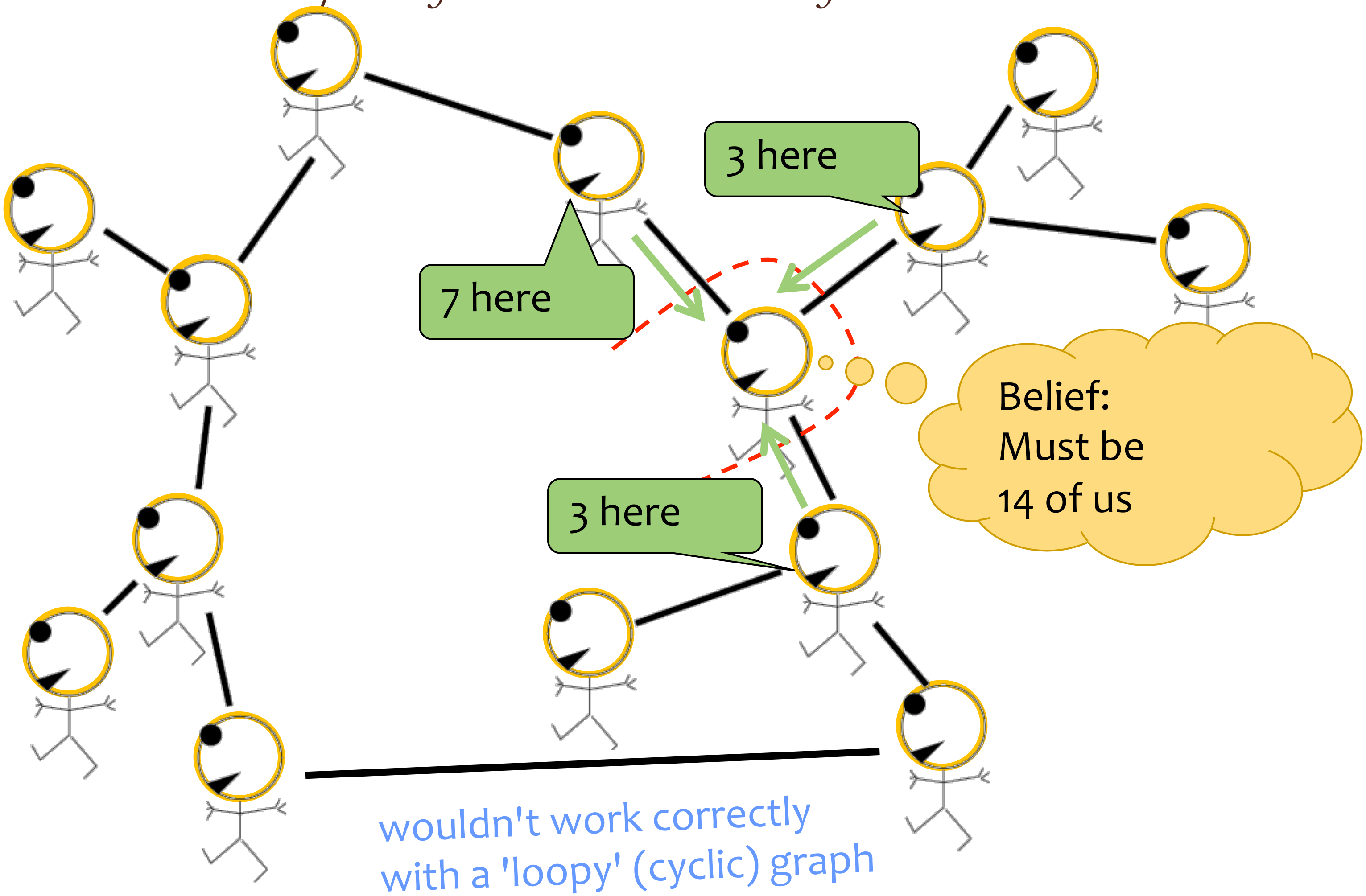
Each soldier receives reports from all branches of tree



adapted from MacKay (2003) textbook

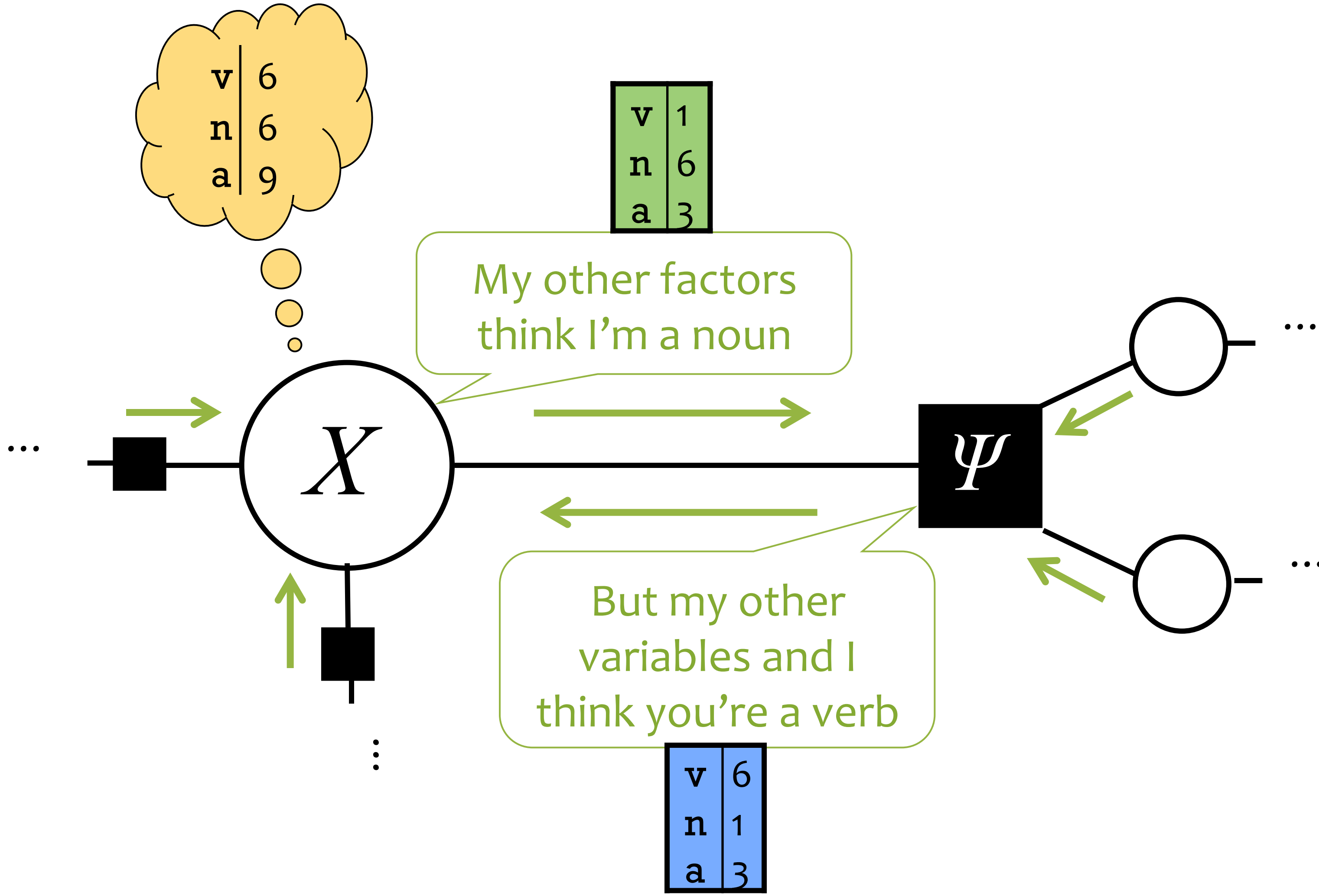
Great Ideas in ML: Message Passing

Each soldier receives reports from all branches of tree



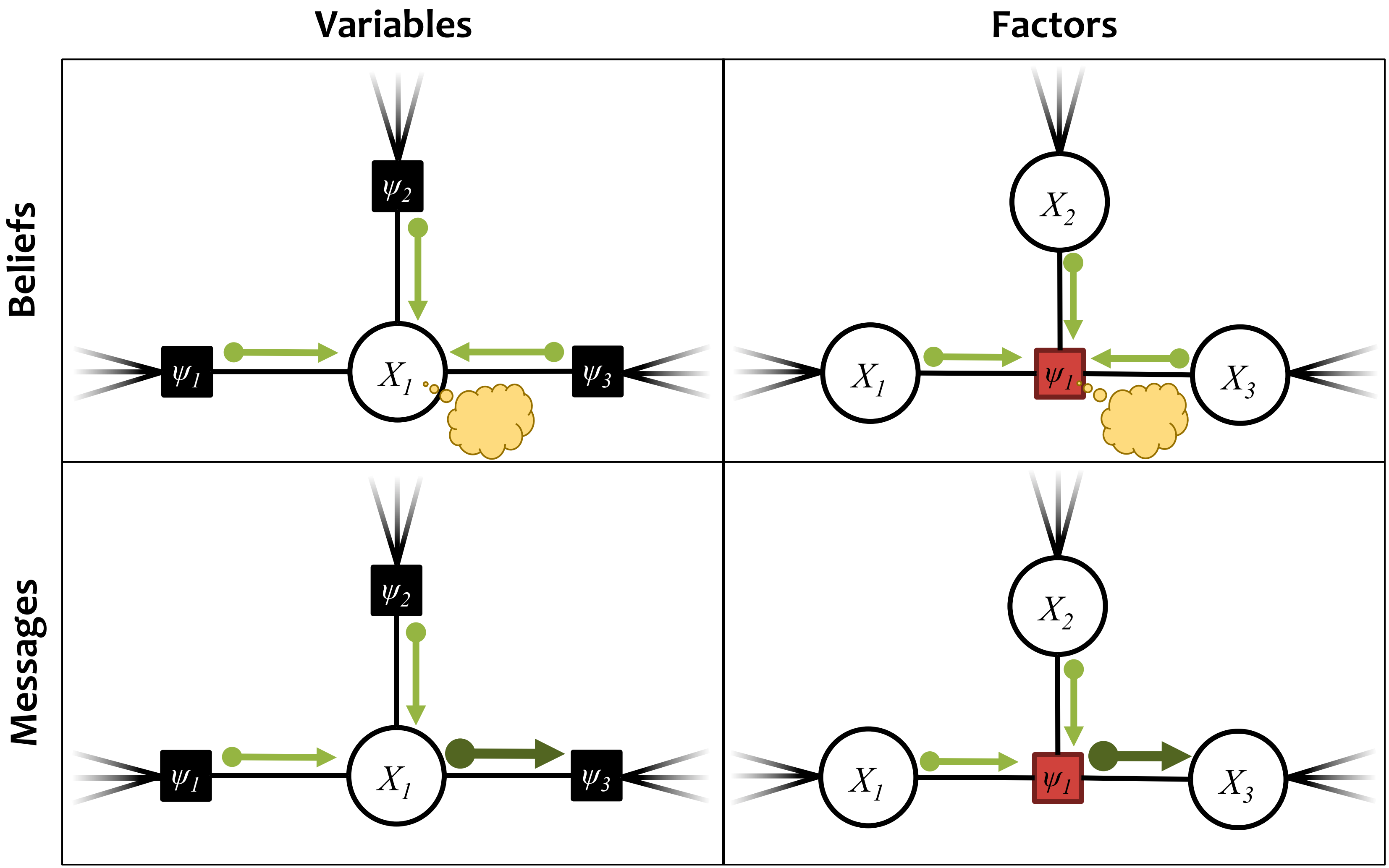
adapted from MacKay (2003) textbook

Message Passing in Belief Propagation



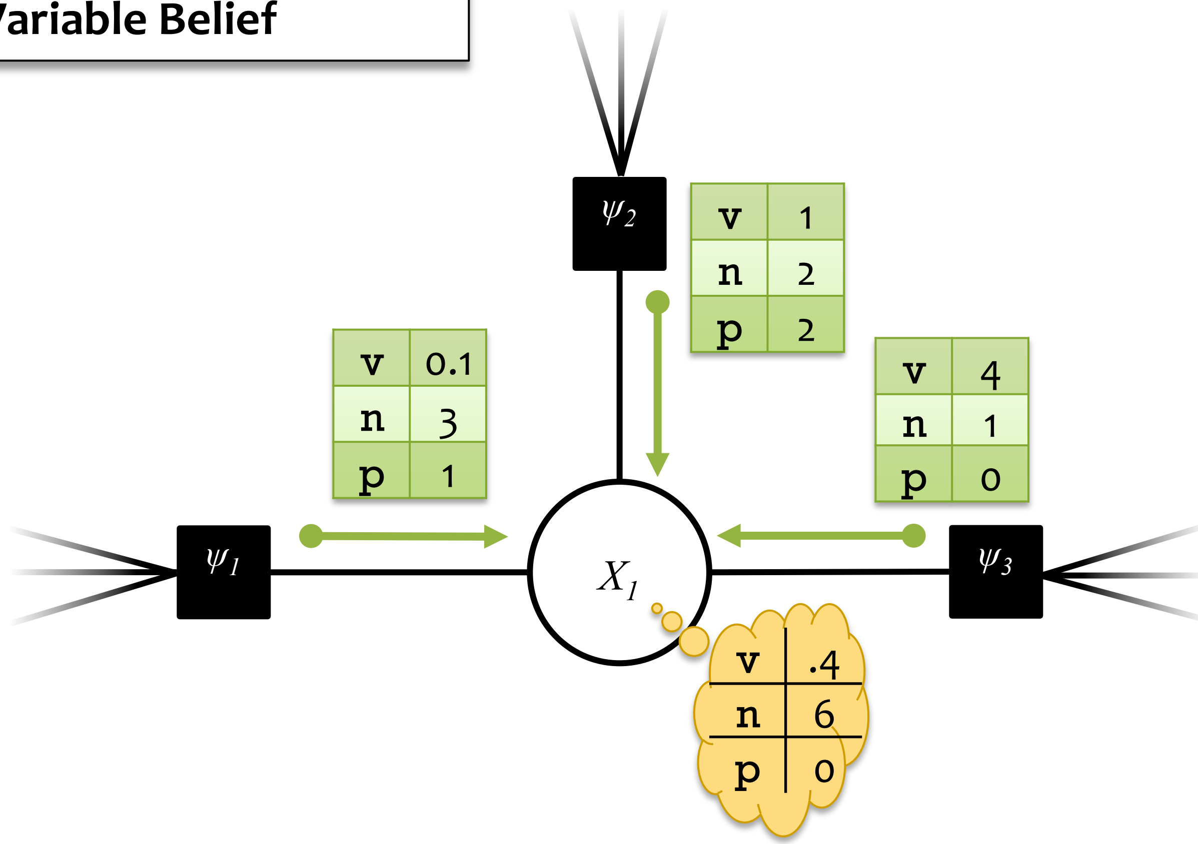
Both of these messages judge the possible values of variable X.
Their product = belief at X = product of all 3 messages to X.

Sum-Product Belief Propagation



Sum-Product Belief Propagation

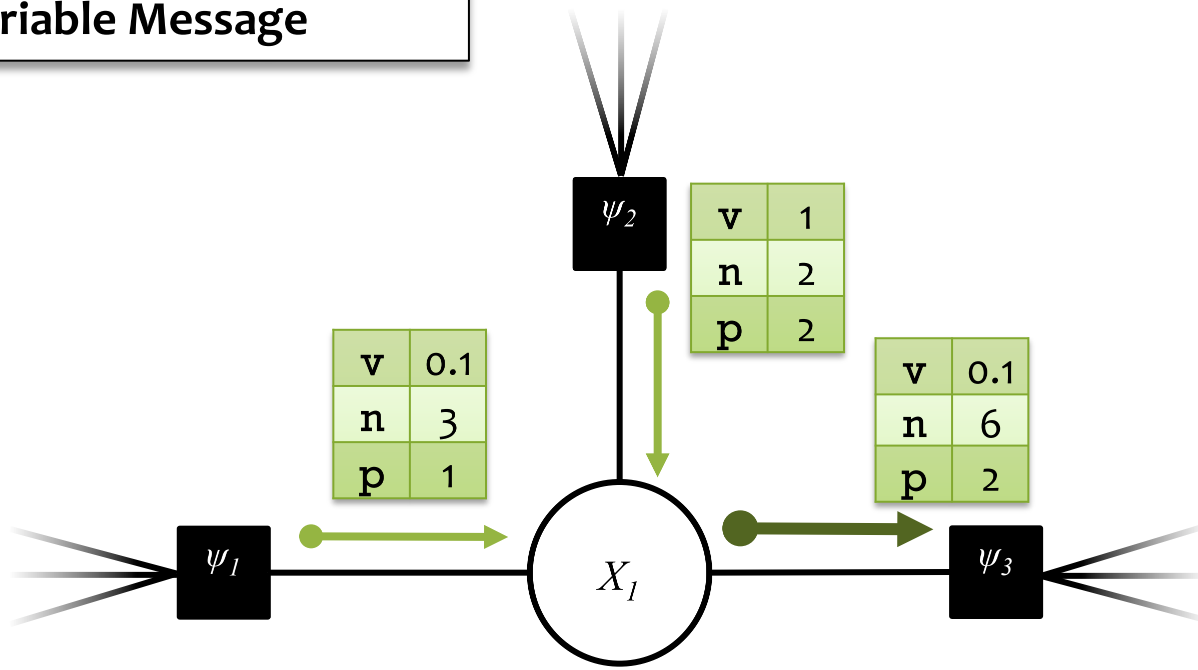
Variable Belief



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

Sum-Product Belief Propagation

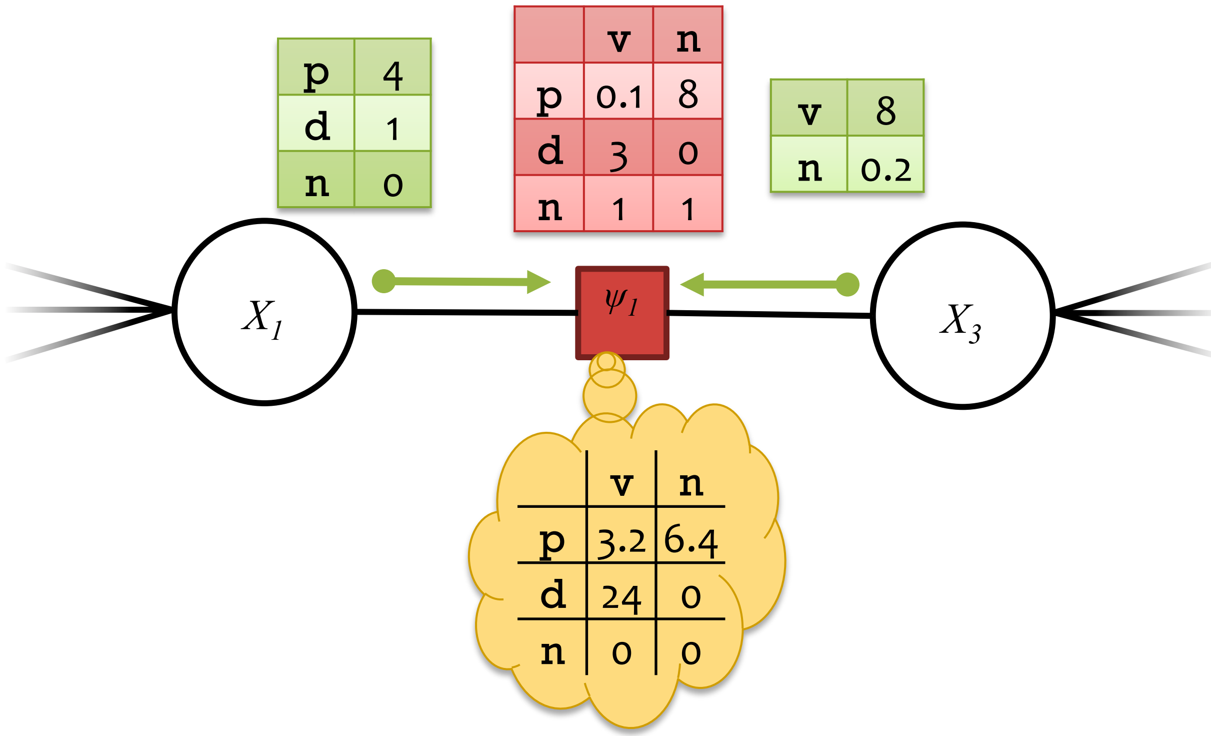
Variable Message



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

Sum-Product Belief Propagation

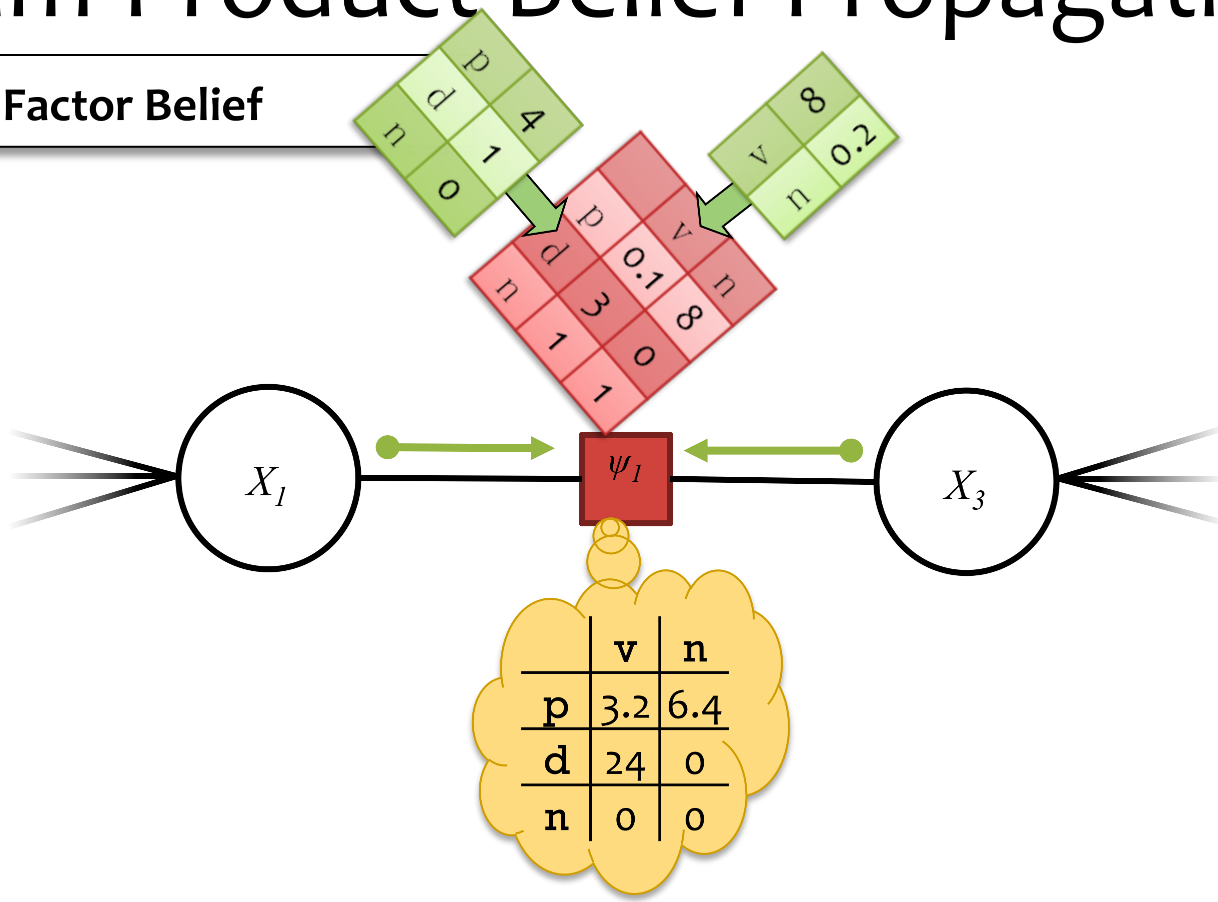
Factor Belief



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

Sum-Product Belief Propagation

Factor Belief

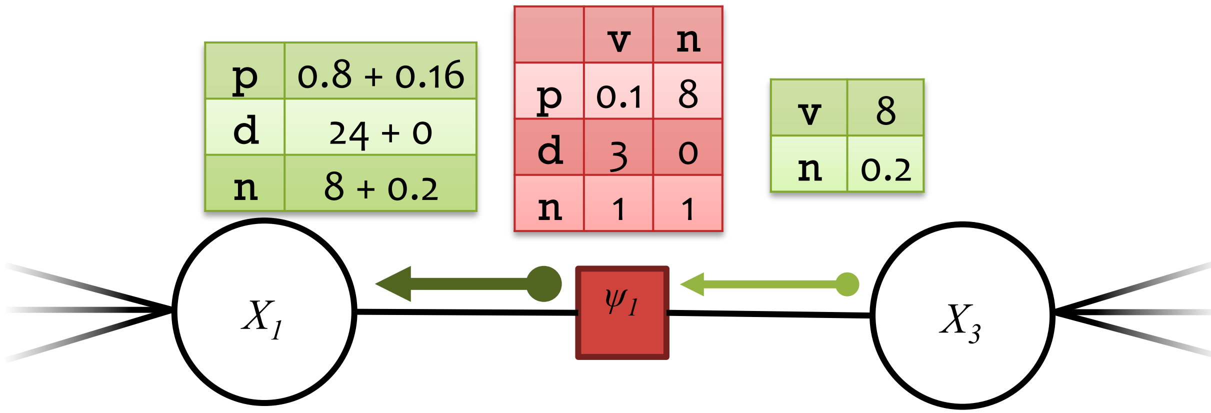


	v	n
p	3.2	6.4
d	24	0
n	0	0

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

Sum-Product Belief Propagation

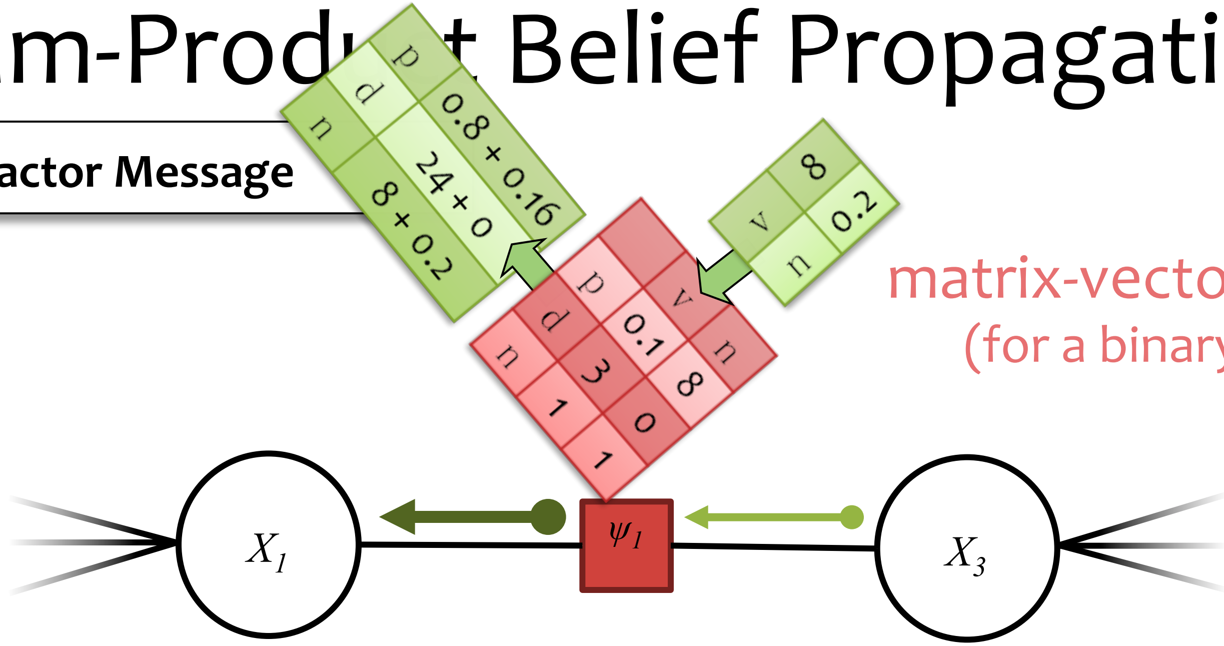
Factor Message



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

Sum-Product Belief Propagation

Factor Message



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.

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

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

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

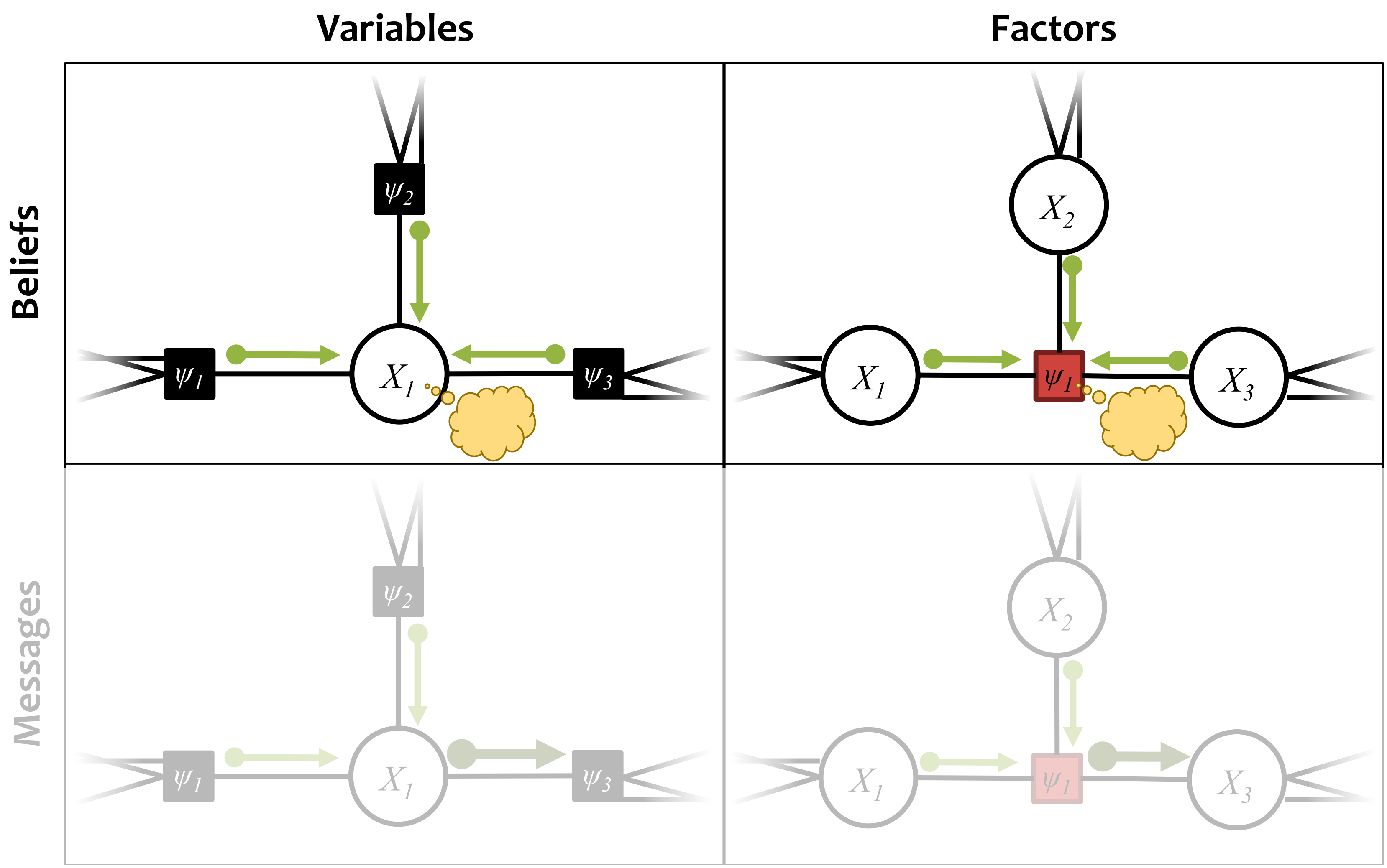
4. Compute the beliefs (unnormalized marginals).

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

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

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

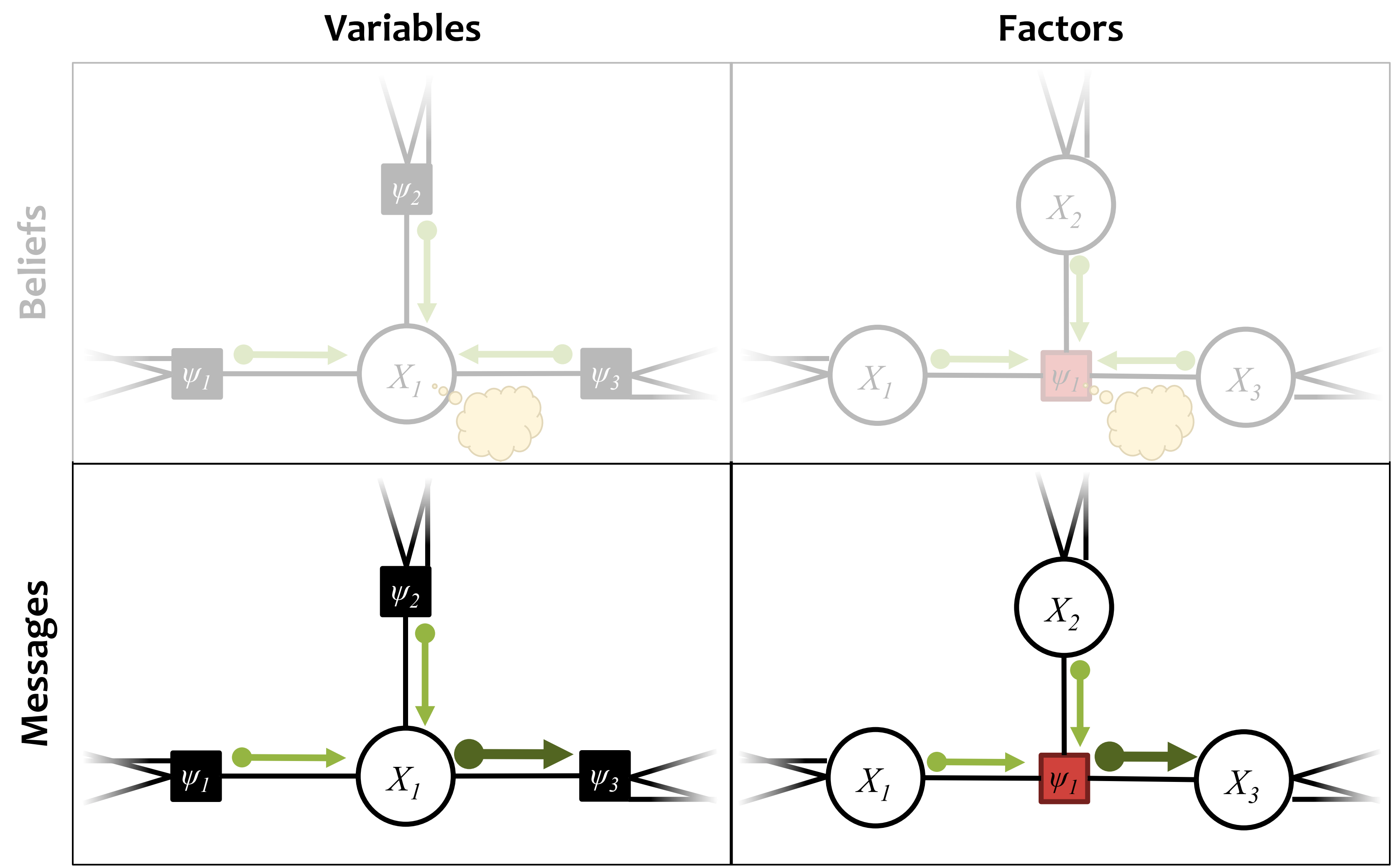
Sum-Product Belief Propagation



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

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

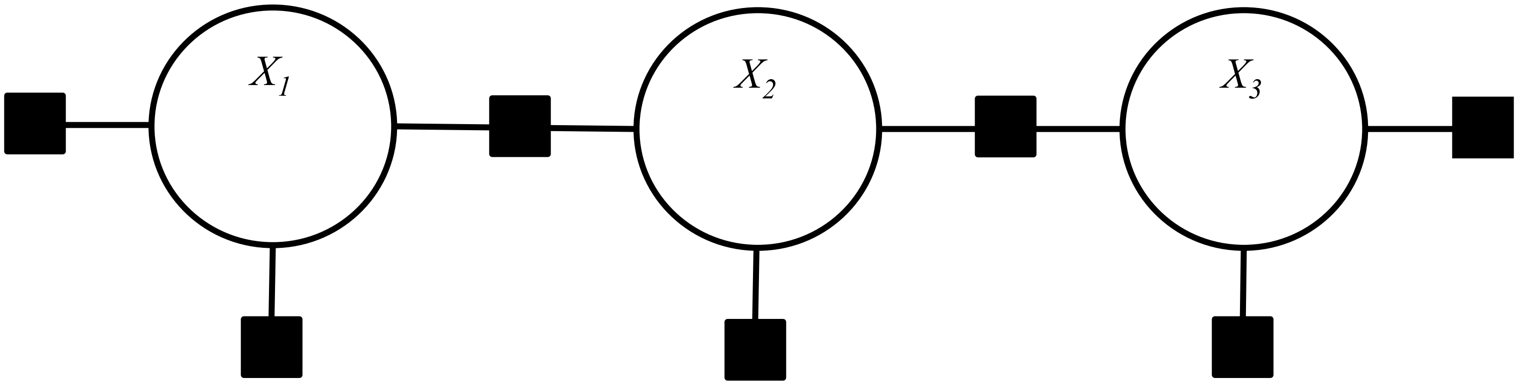
Sum-Product Belief Propagation



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

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

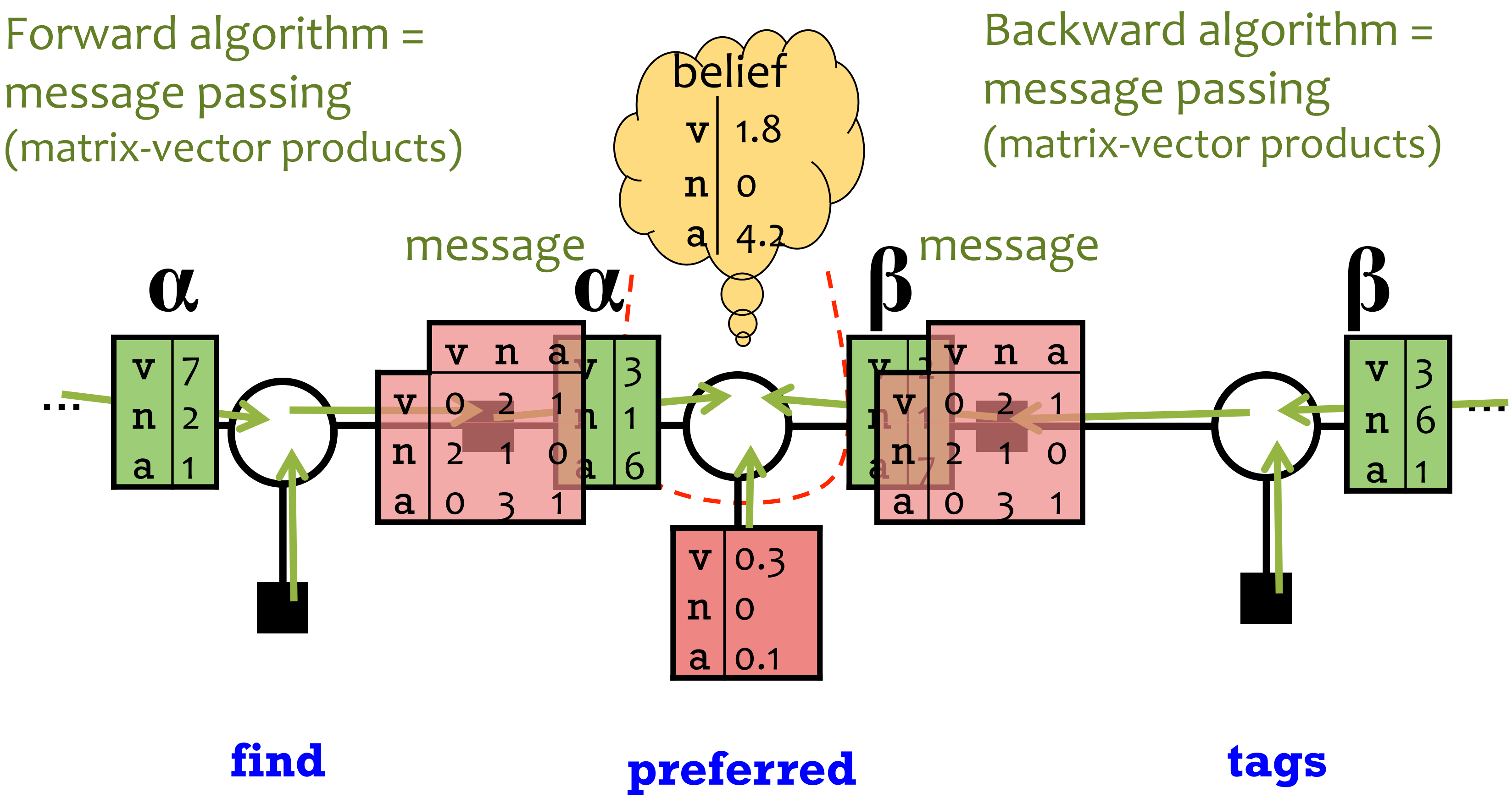
CRF Tagging Model



find | preferred | tags

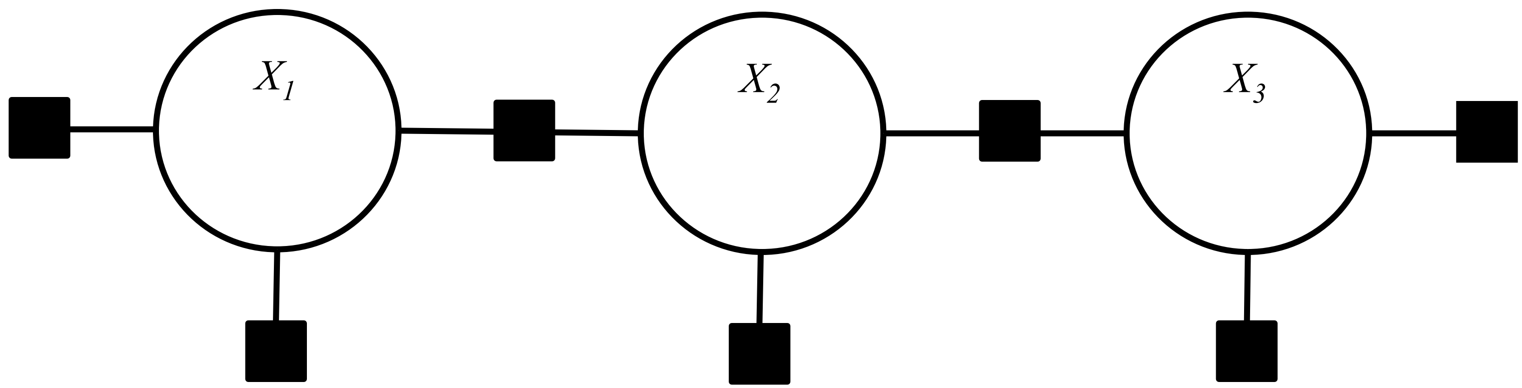
Could be verb or noun | *Could be adjective or verb* | *Could be noun or verb*

CRF Tagging by Belief Propagation



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

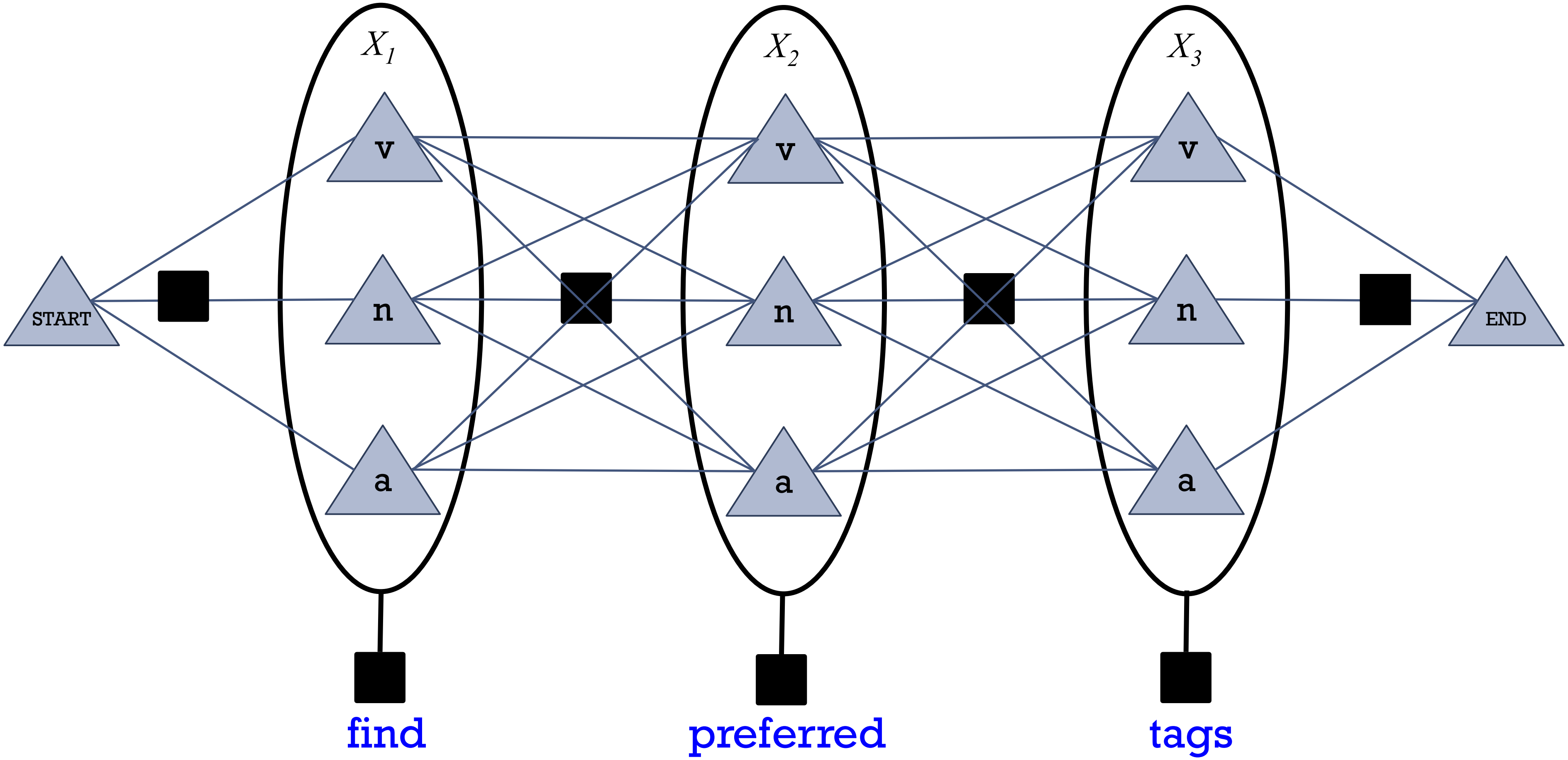
So Let's Review Forward-Backward …



find | preferred | tags

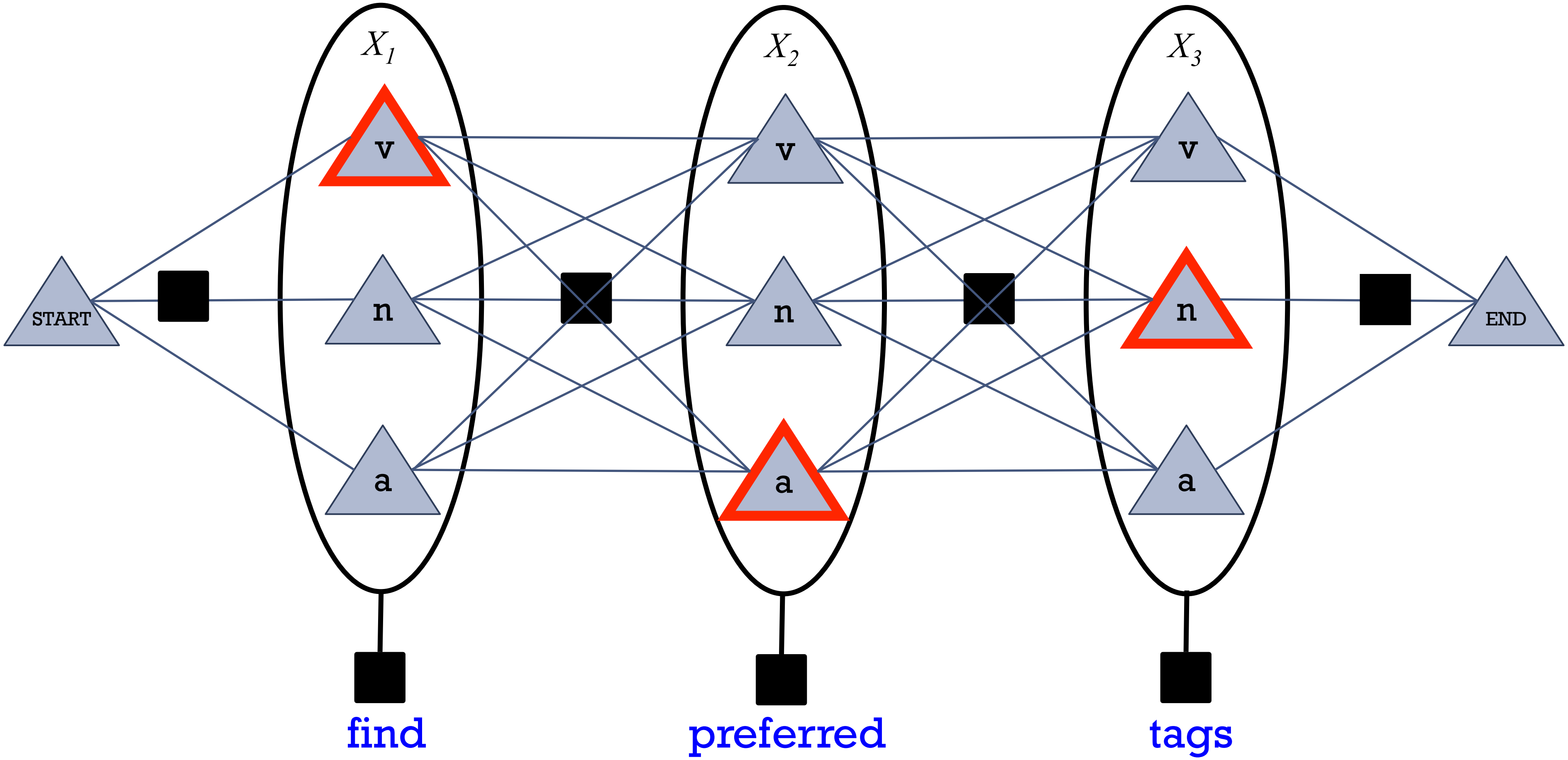
Could be verb or noun | *Could be adjective or verb* | *Could be noun or verb*

So Let's Review Forward-Backward …



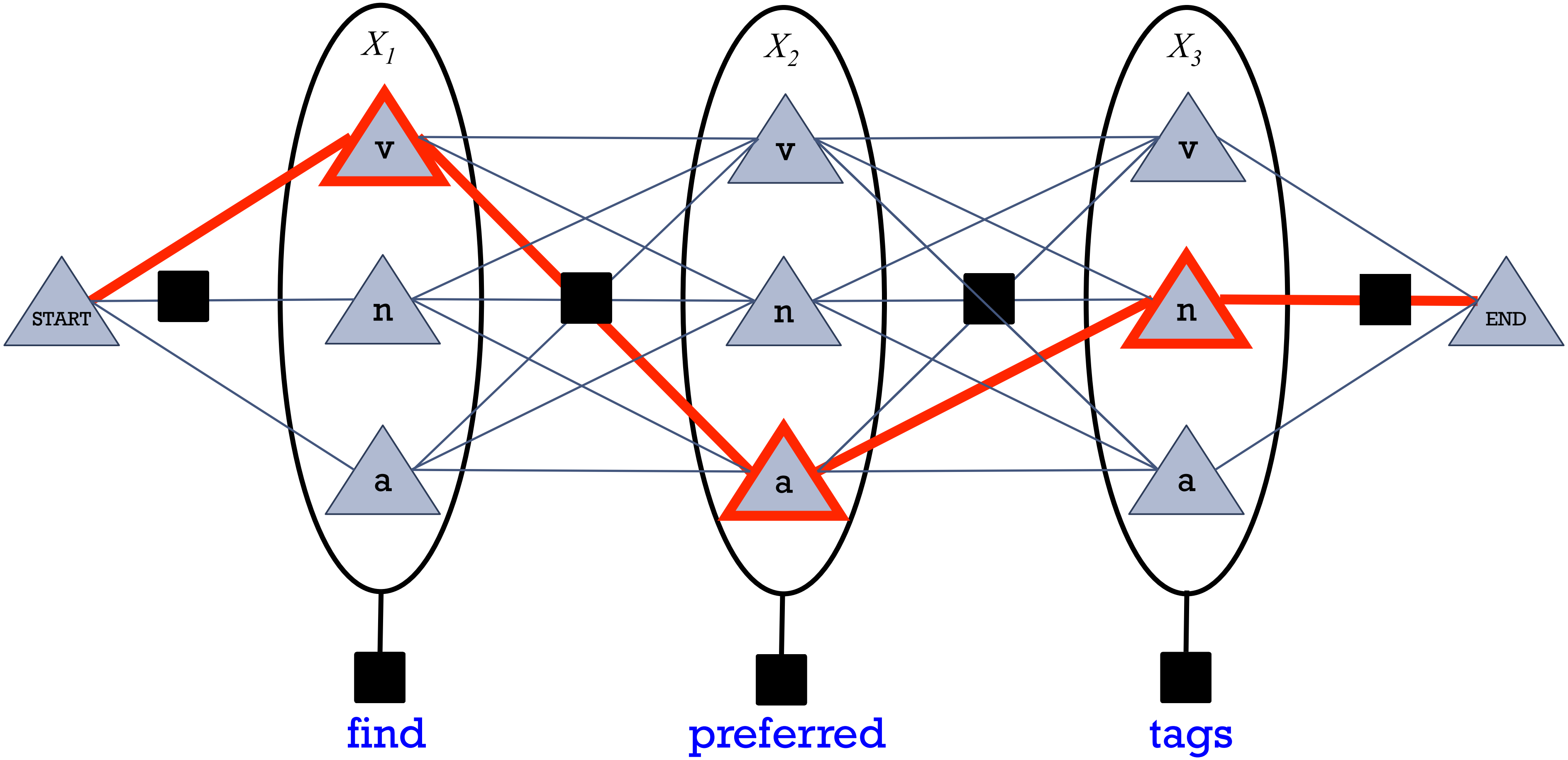
- Show the possible *values* for each variable

So Let's Review Forward-Backward …



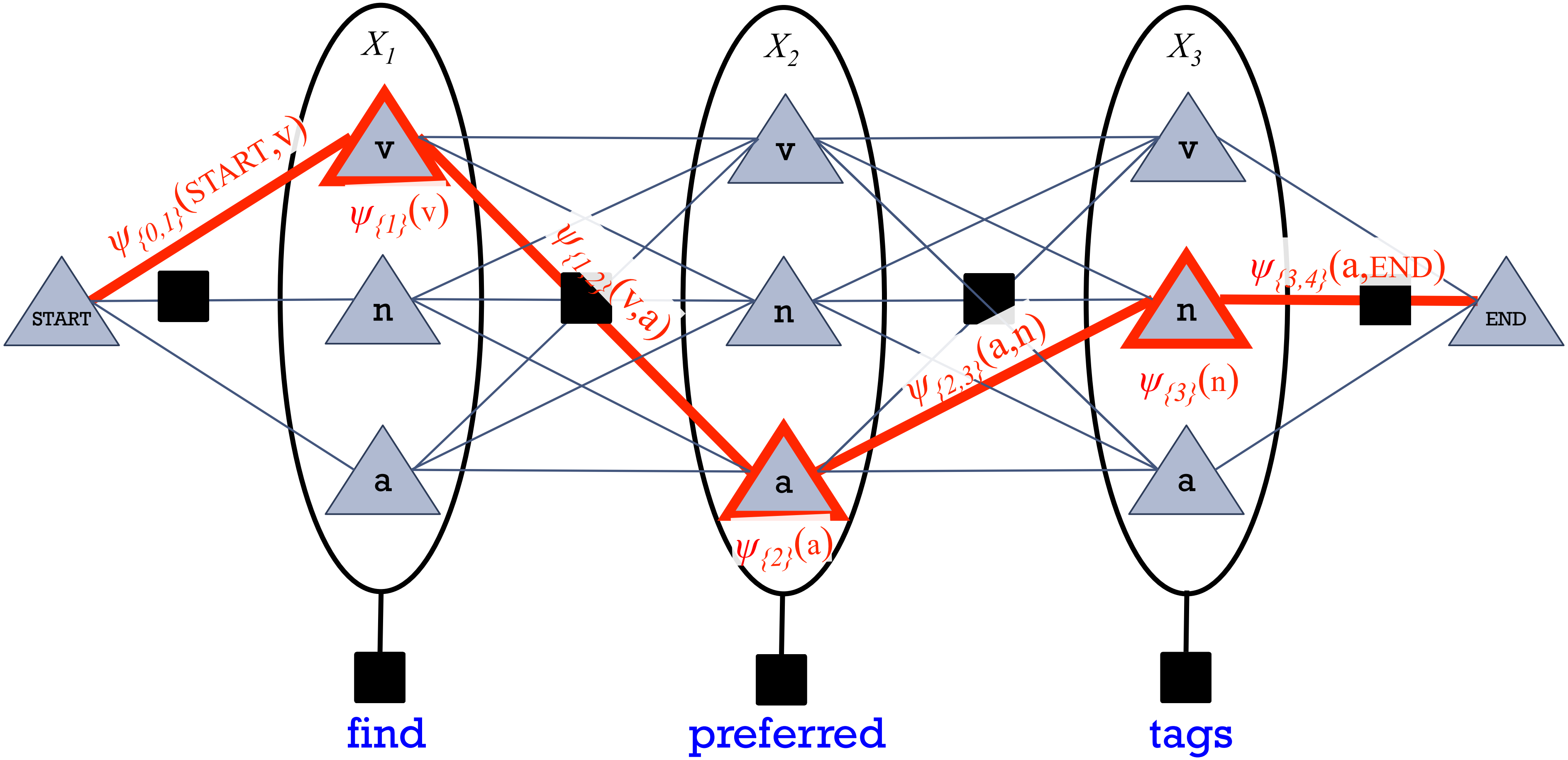
- Let's show the possible *values* for each variable
- One possible assignment

So Let's Review Forward-Backward …



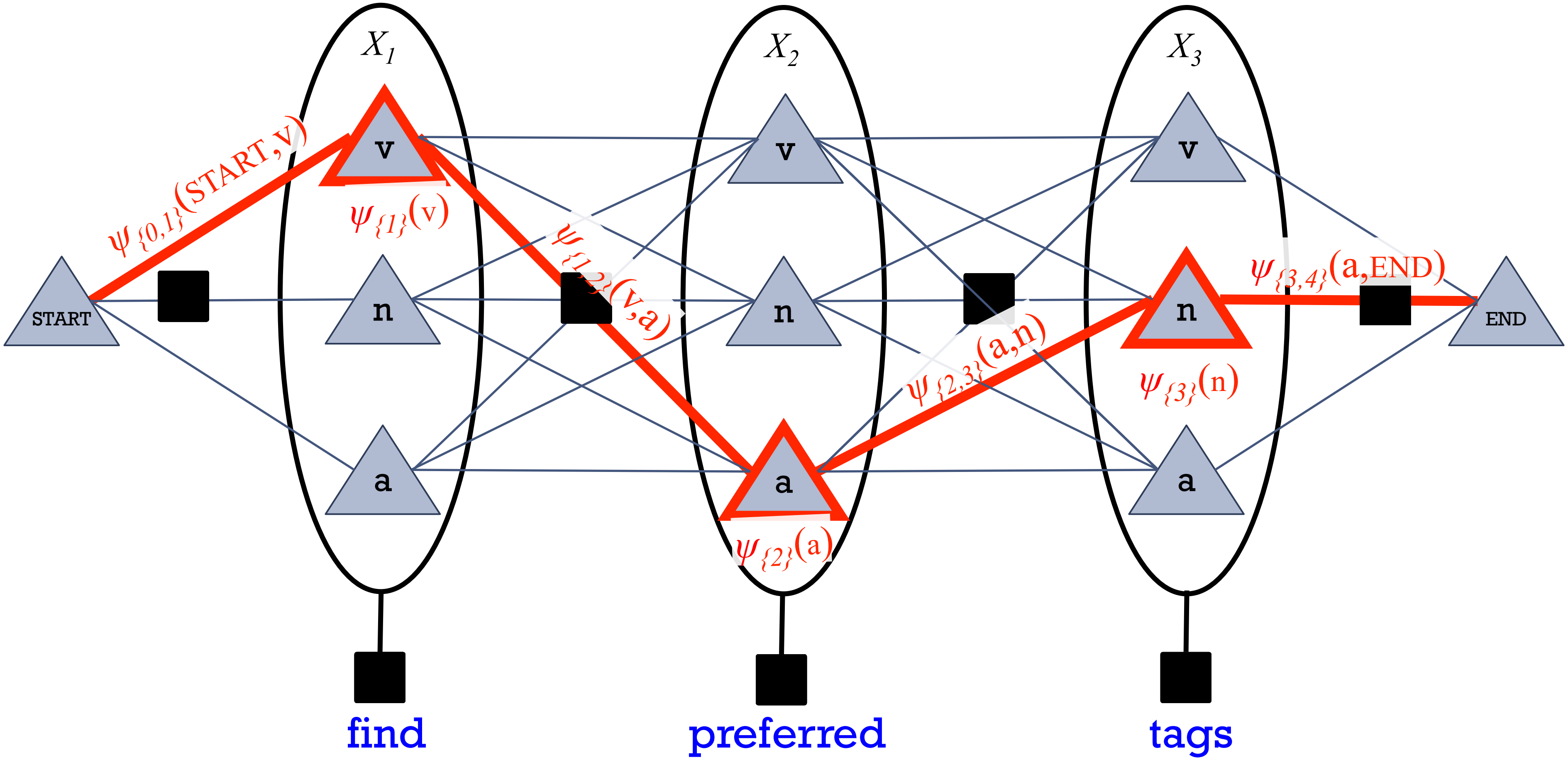
- Let's show the possible *values* for each variable
- One possible assignment
- And what the 7 factors think of it …

Viterbi Algorithm: Most Probable Assignment



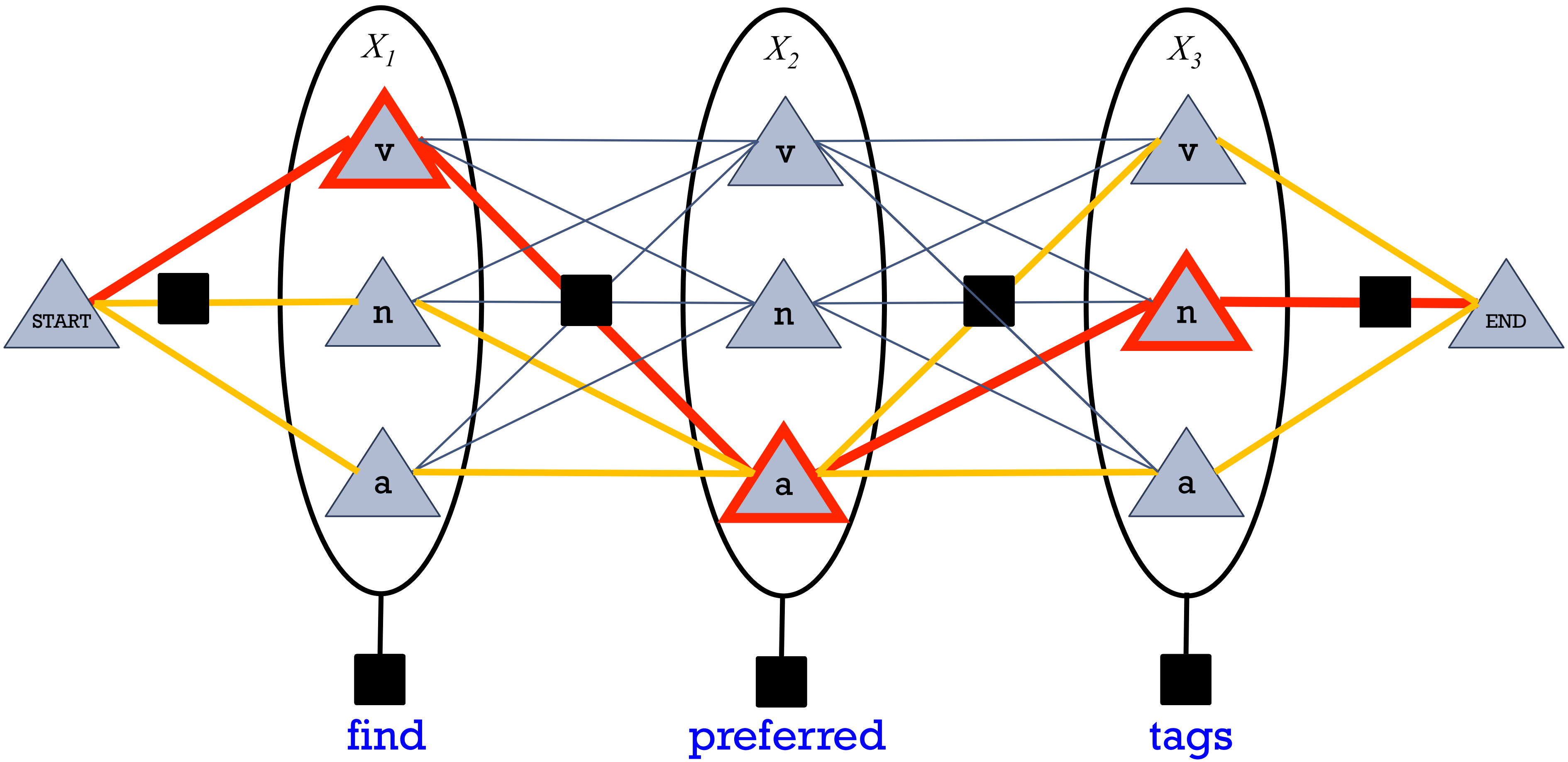
- So p(**v a n**) = (1/Z) * product of 7 numbers
- Numbers associated with edges and nodes of path
- Most probable assignment = **path with highest product**

Viterbi Algorithm: Most Probable Assignment



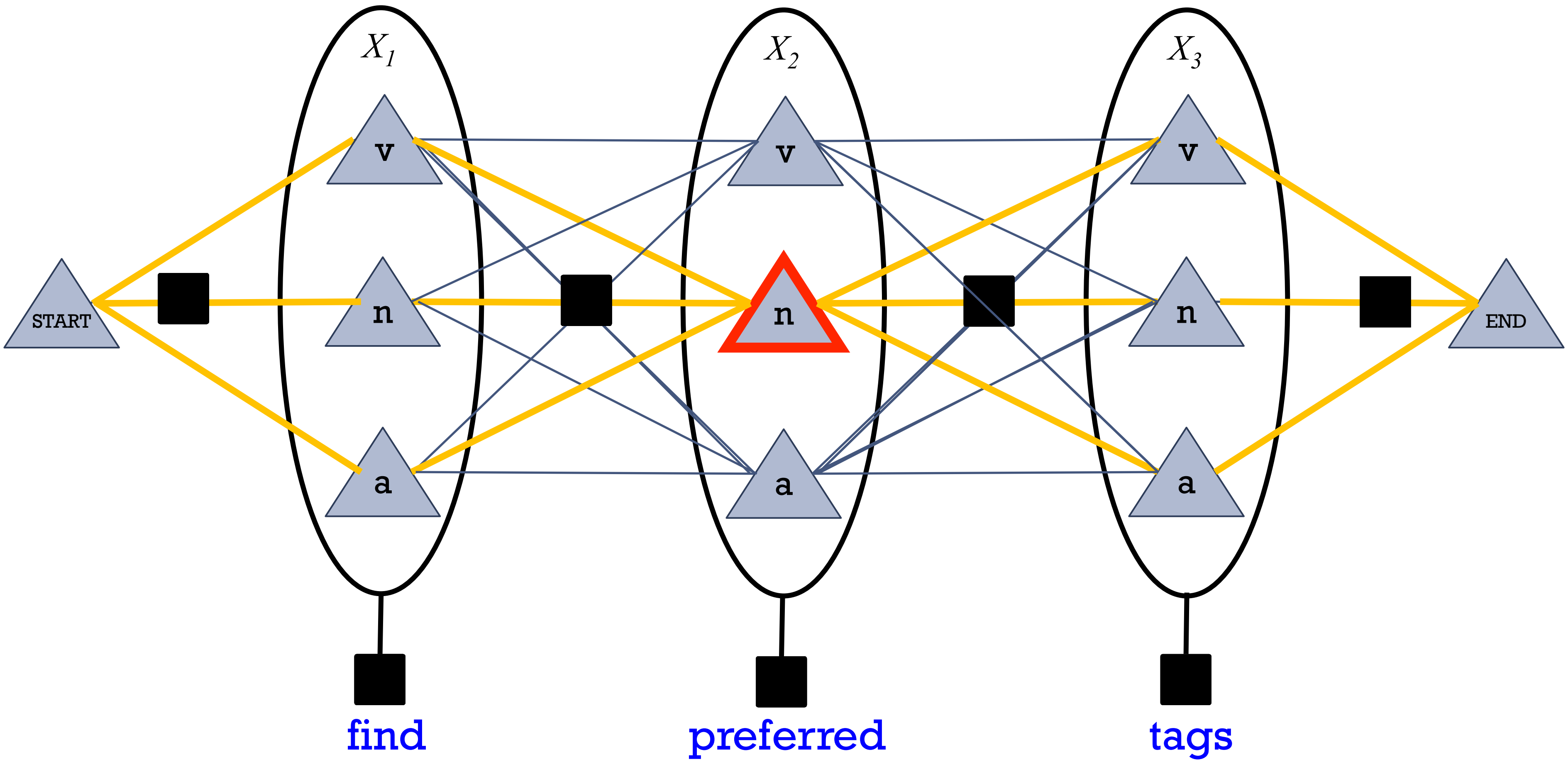
- So p(**v a n**) = (1/Z) * product weight of one path

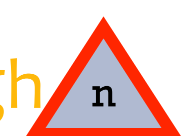
Forward-Backward Algorithm: Finds Marginals



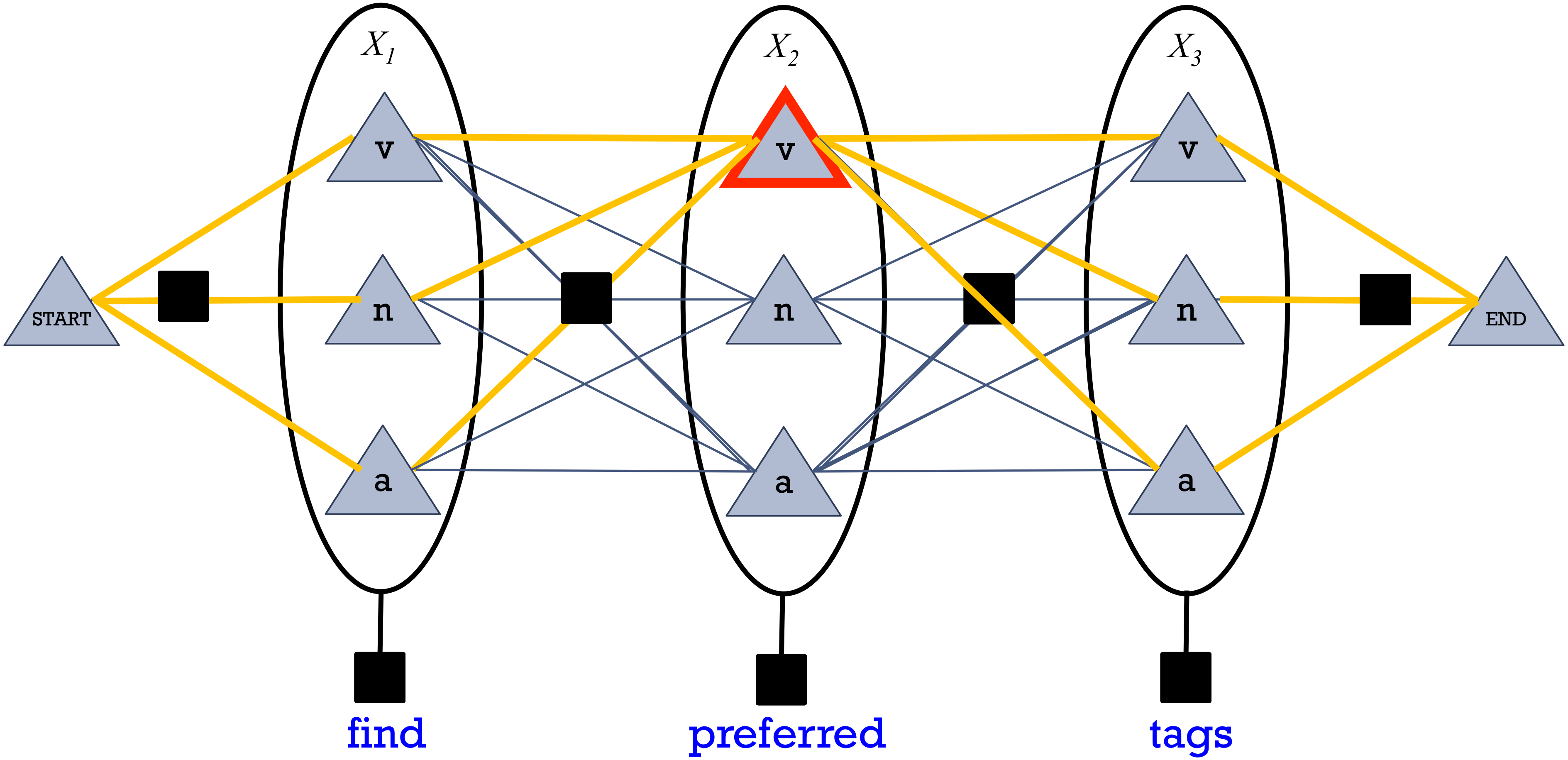
- So p(**v a n**) = (1/Z) * product weight of one path
- Marginal probability $p(X_2 = a)$
  = (1/Z) * total weight of *all* paths through a

Forward-Backward Algorithm: Finds Marginals



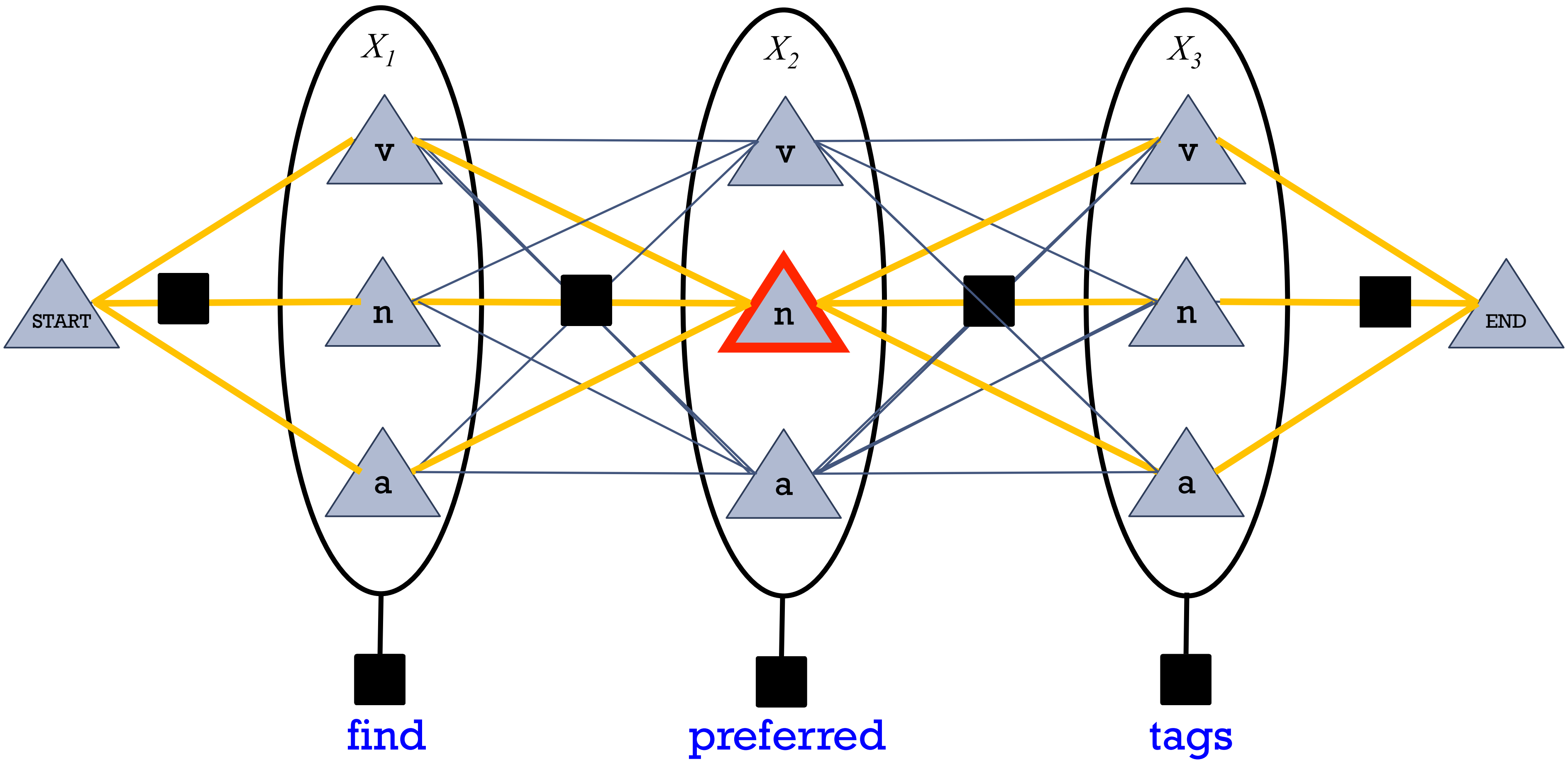
- So p(**v a n**) = (1/Z) * product weight of one path
- Marginal probability p(X_2 = a)
  = (1/Z) * total weight of *all* paths through 

Forward-Backward Algorithm: Finds Marginals



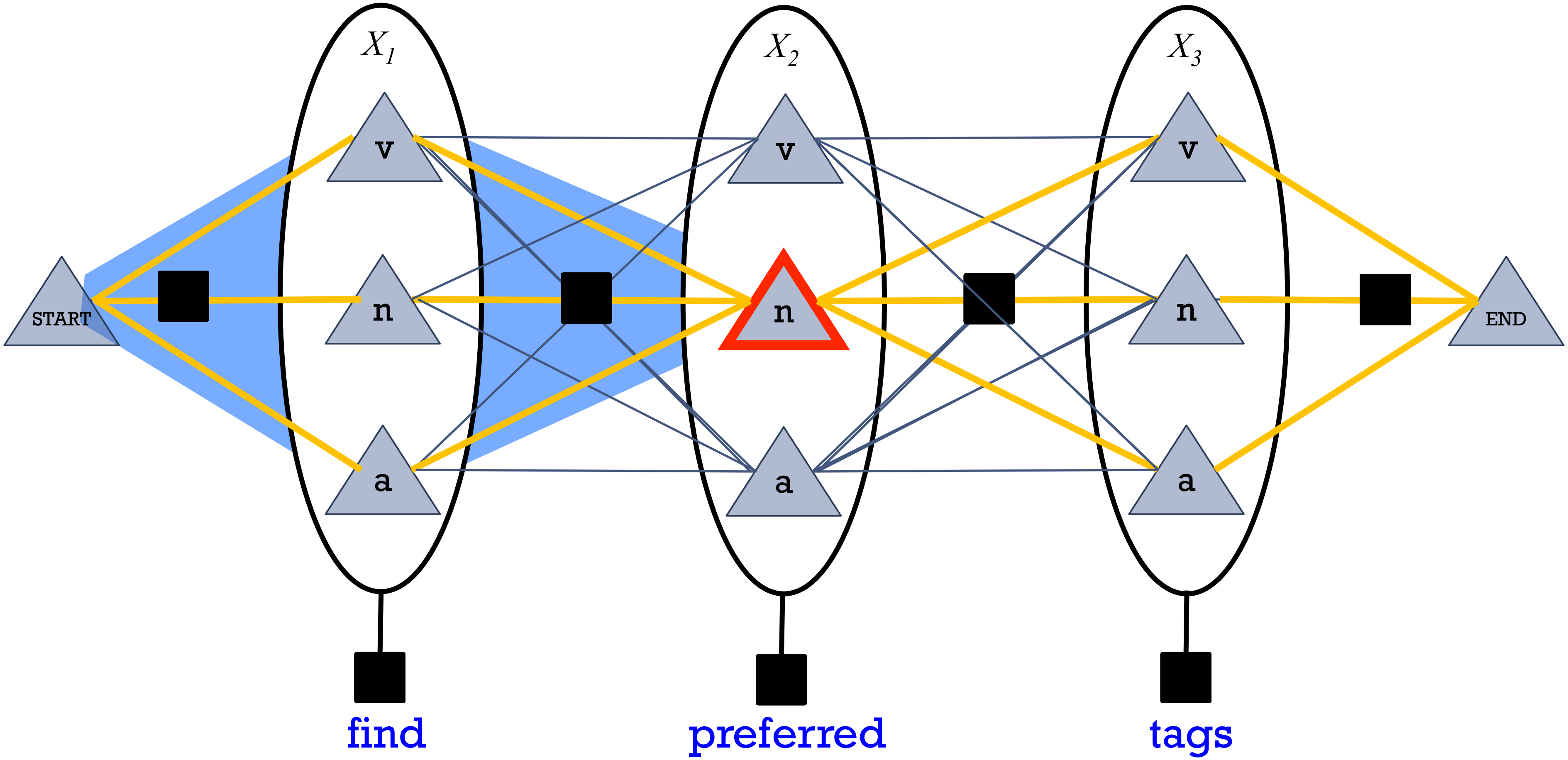
- So p(**v a n**) = (1/Z) * product weight of one path
- Marginal probability $p(X_2 = a)$
  = (1/Z) * total weight of *all* paths through **v**

Forward-Backward Algorithm: Finds Marginals



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

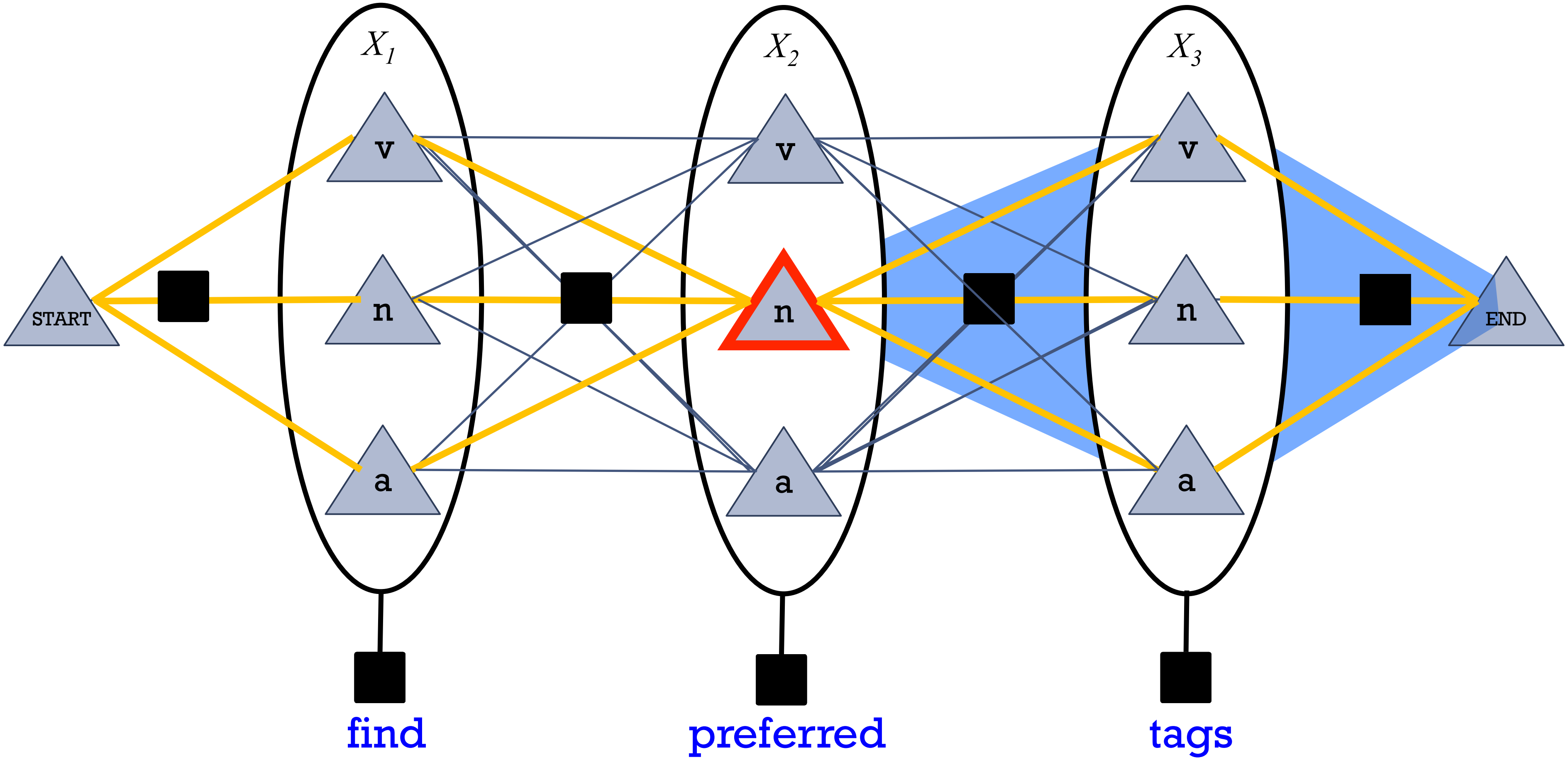
Forward-Backward Algorithm: Finds Marginals



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

(found by dynamic programming: matrix-vector products)

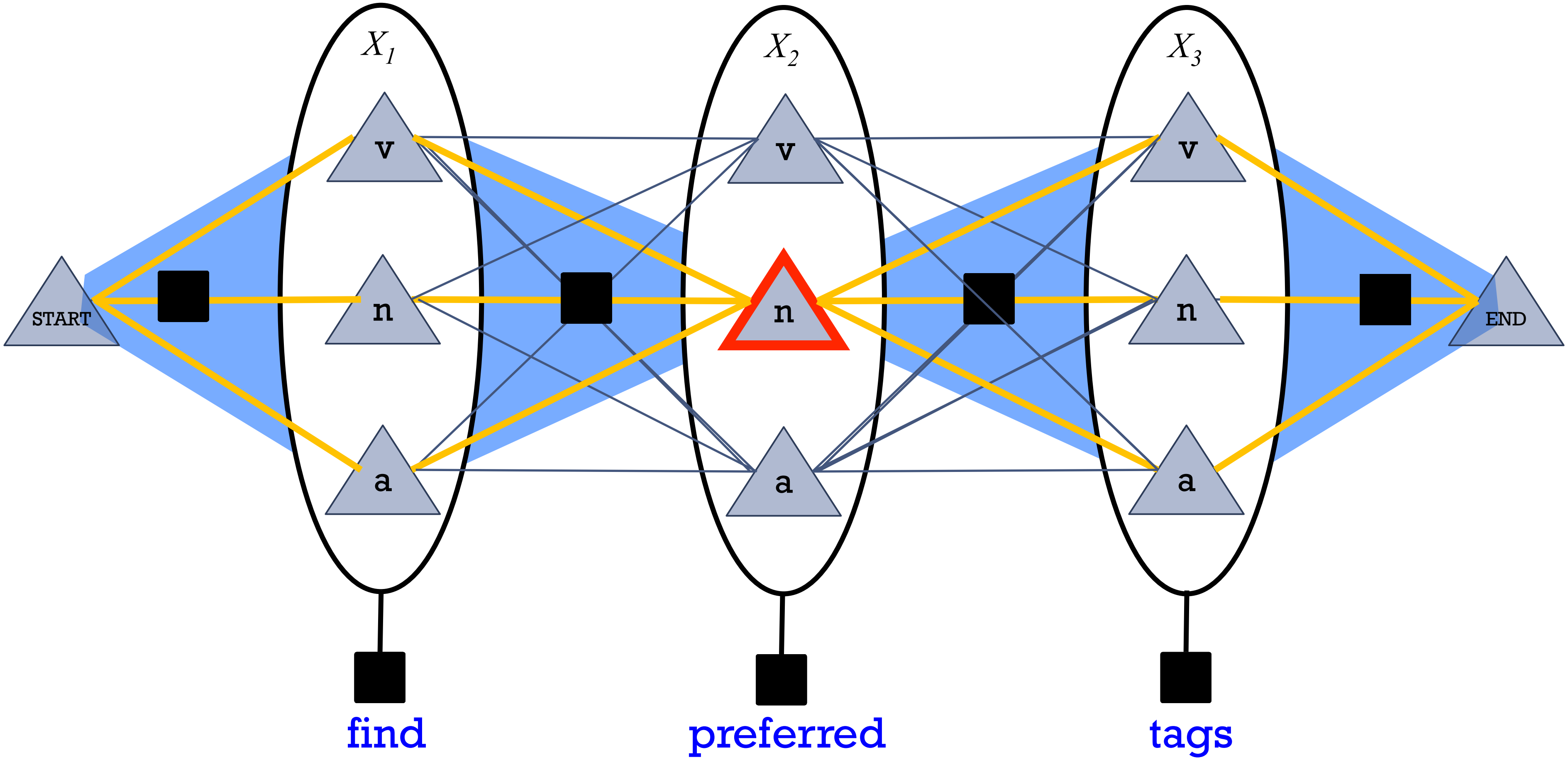
Forward-Backward Algorithm: Finds Marginals



$\beta_2(\mathbf{n})$ = 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)

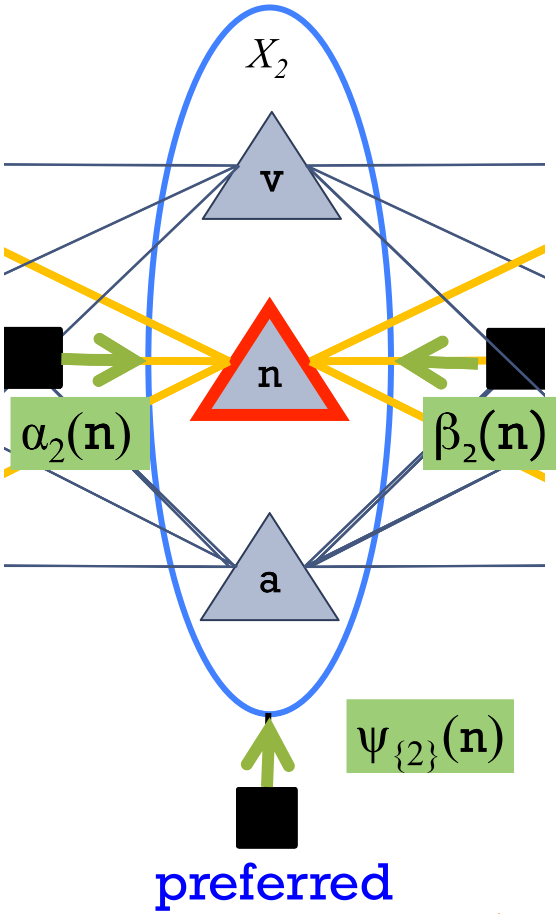
$\beta_2(\mathbf{n})$ = total weight of these path *suffixes* (x + y + z)

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.

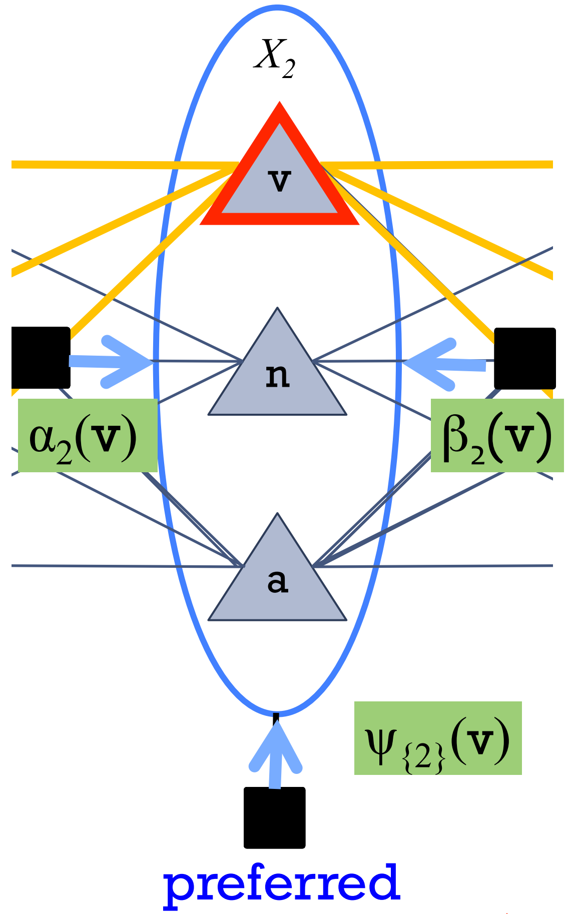


"belief that X_2 = **n**"

total weight of *all* paths through **n**

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

Forward-Backward Algorithm: Finds Marginals



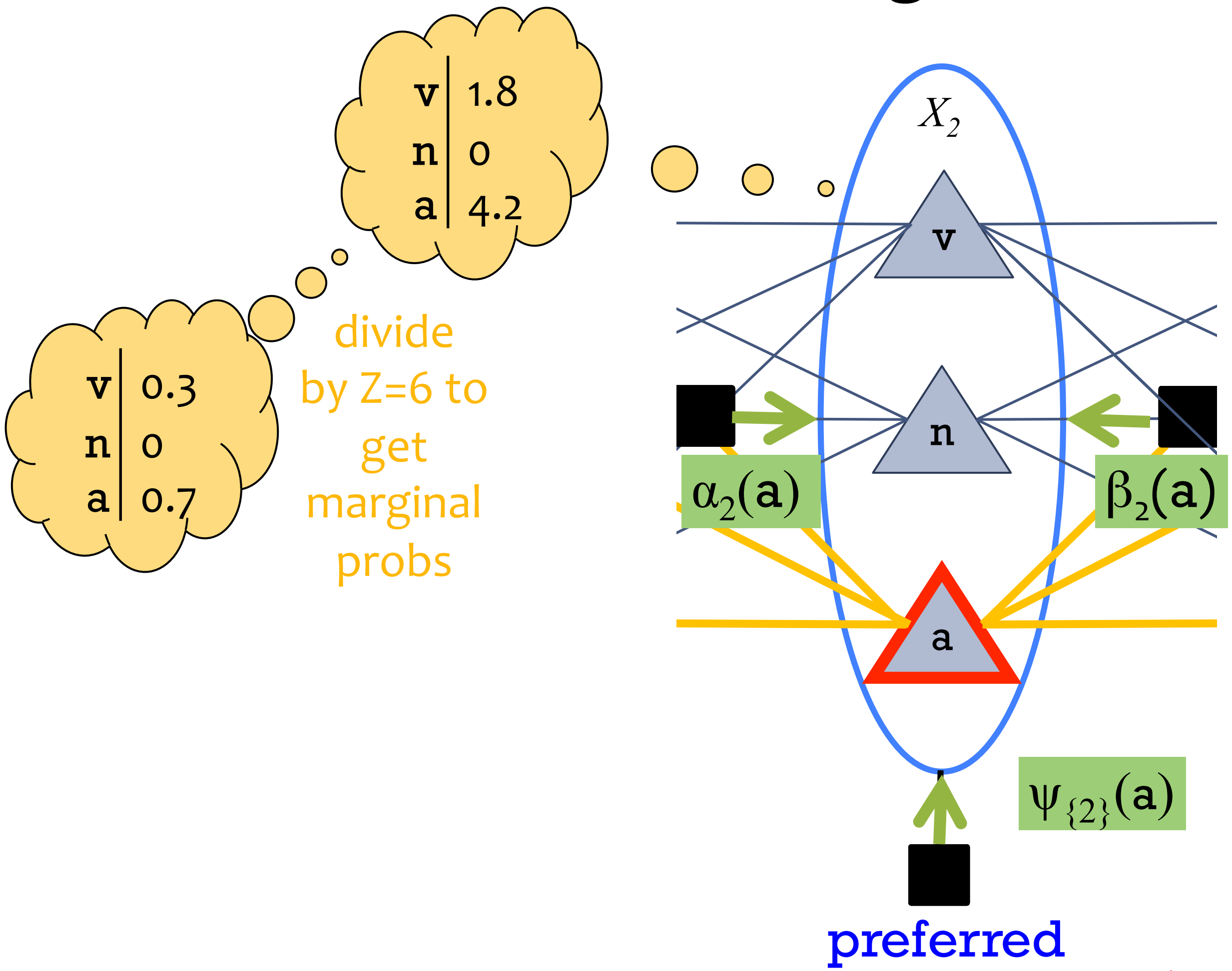
"belief that $X_2 = \mathbf{v}$"

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

total weight of *all* paths through v

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

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)

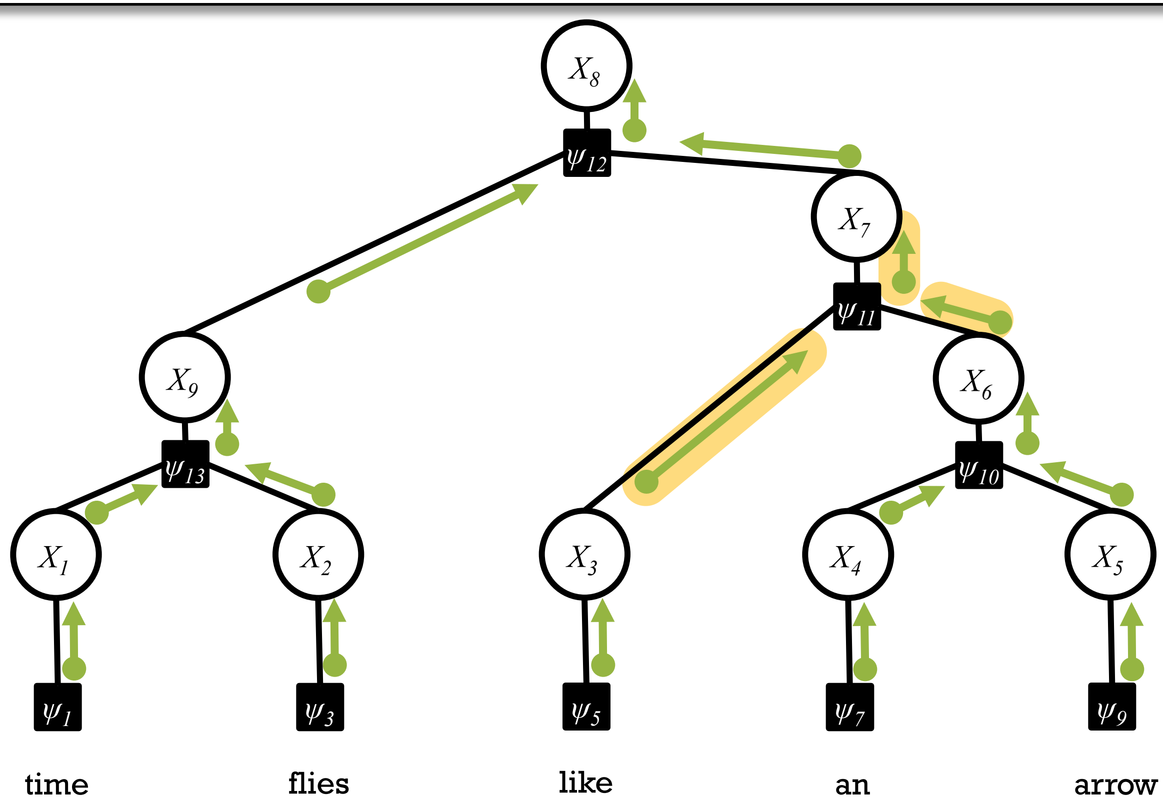
total weight of *all* paths through a

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

(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.

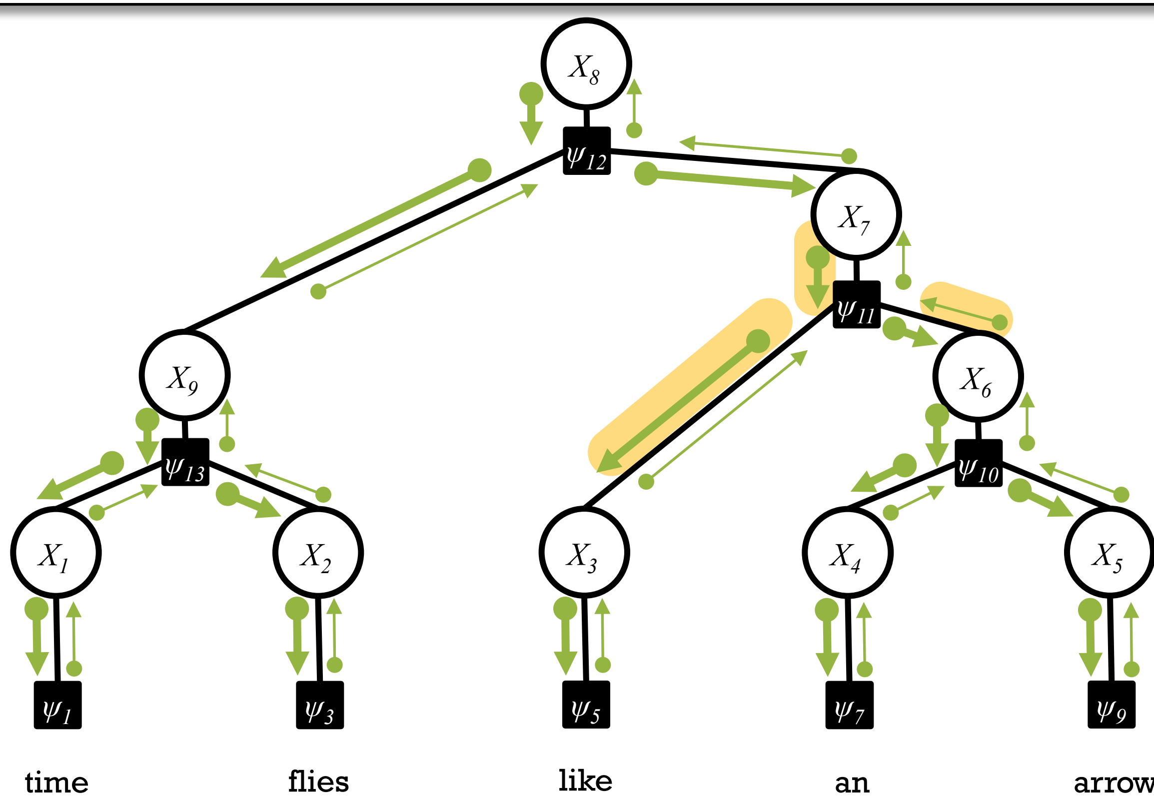


(Acyclic) Belief Propagation

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**.

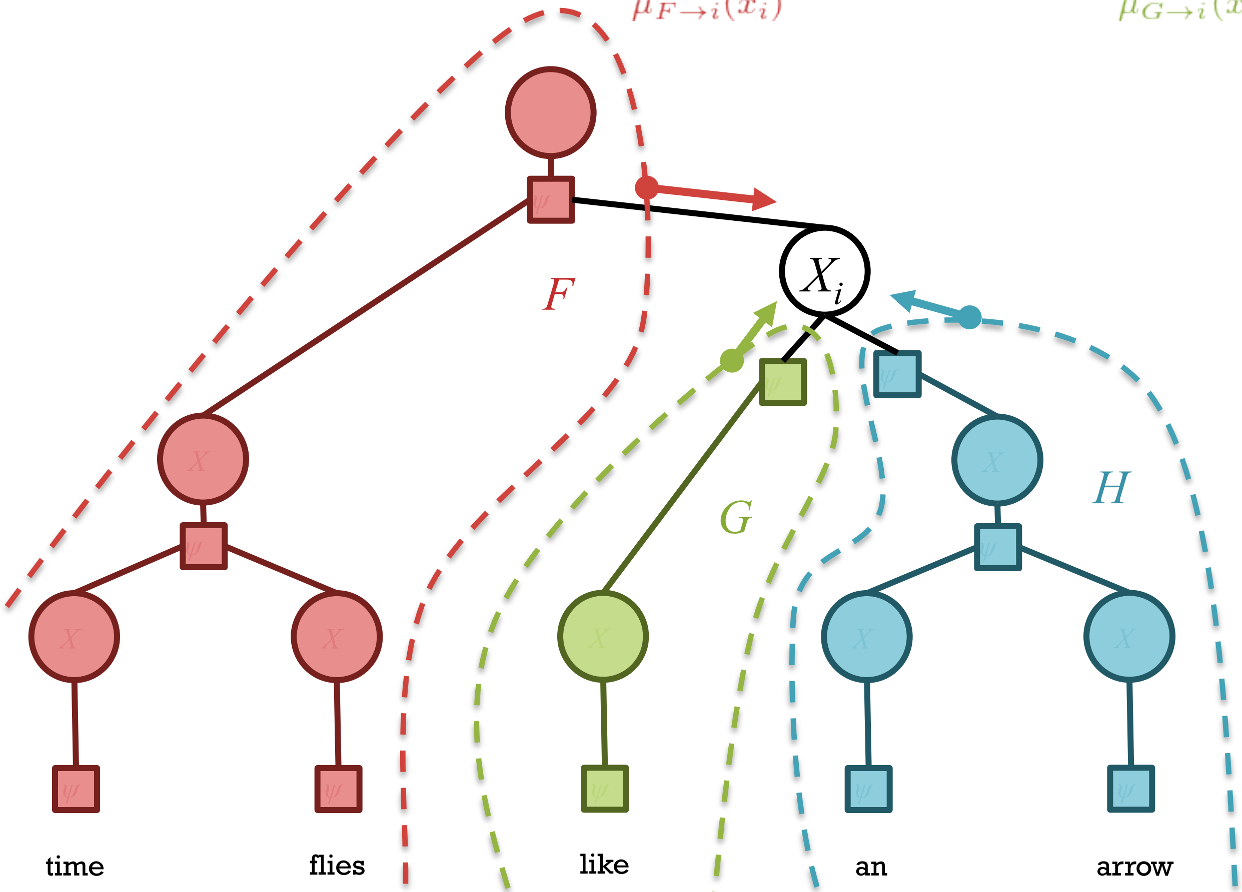
A node computes an outgoing message along an edge
only after it has received incoming messages along all its other edges.



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)

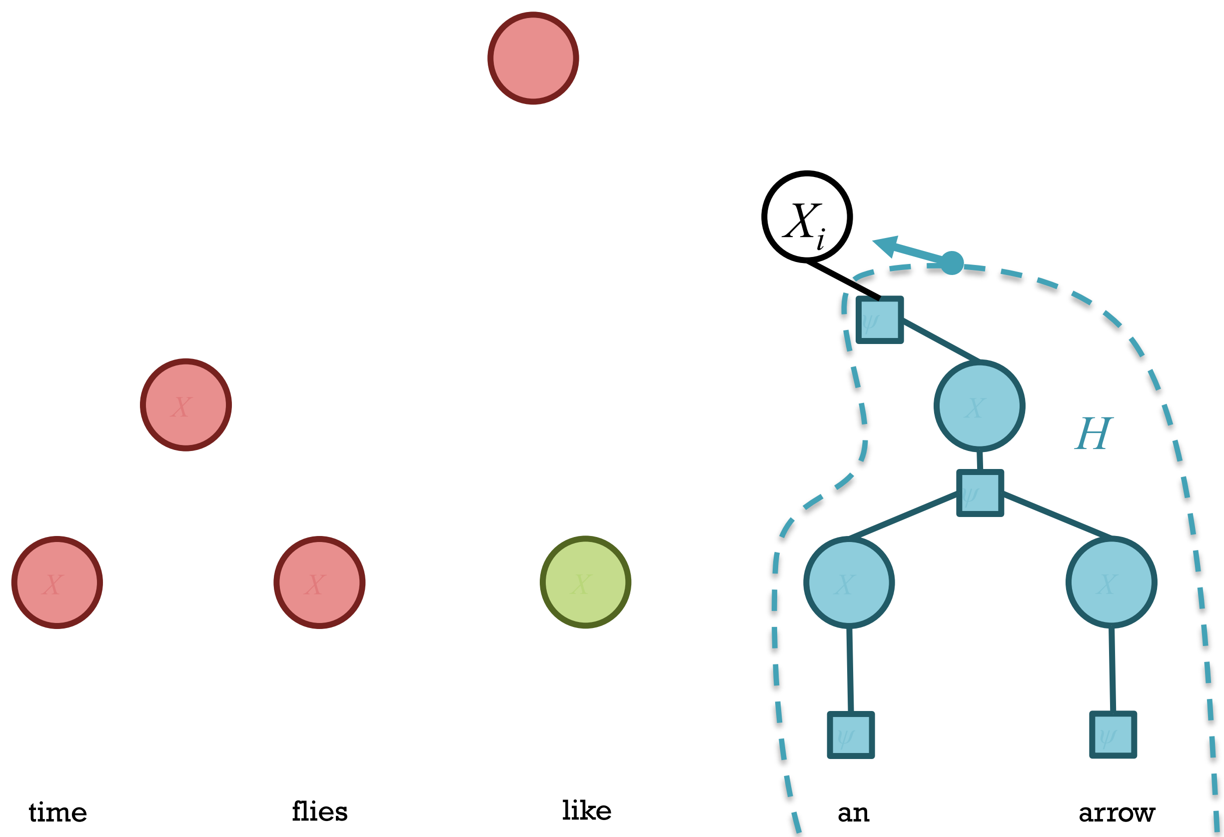
Acyclic BP as Dynamic Programming

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

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

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

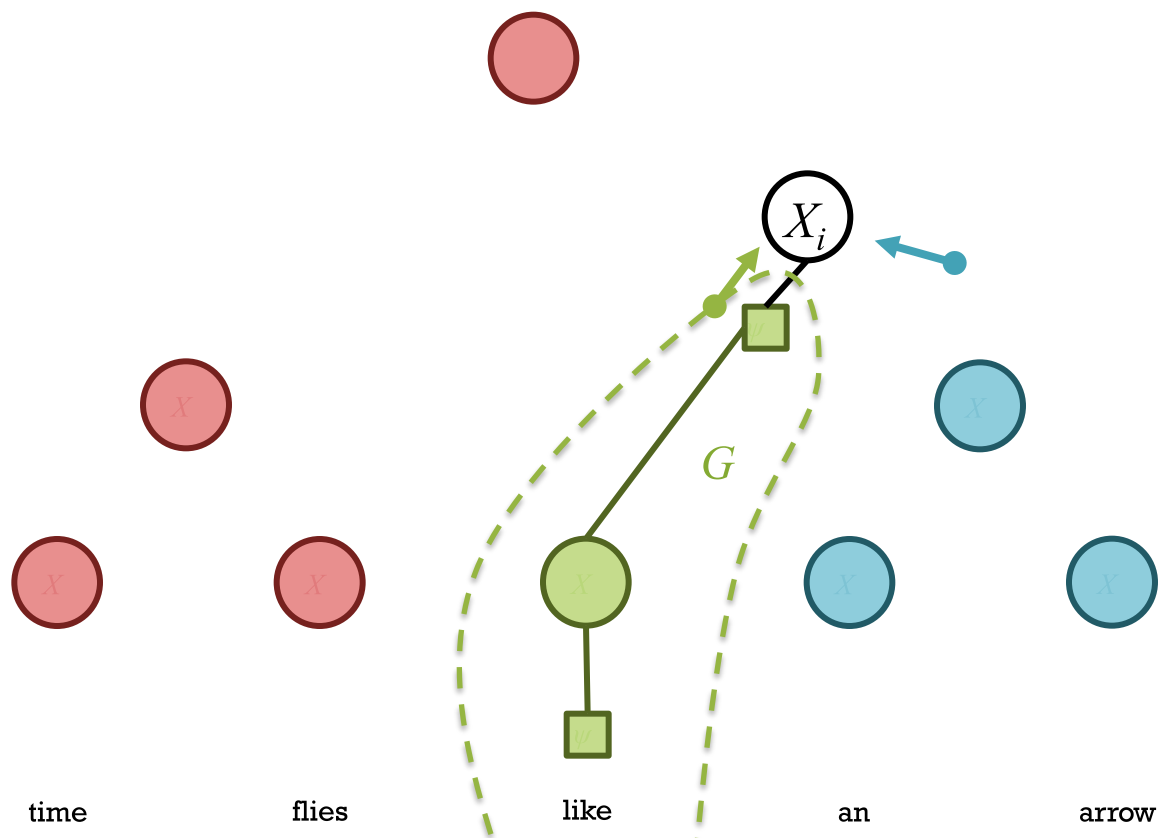


Message ***to*** *a variable*

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

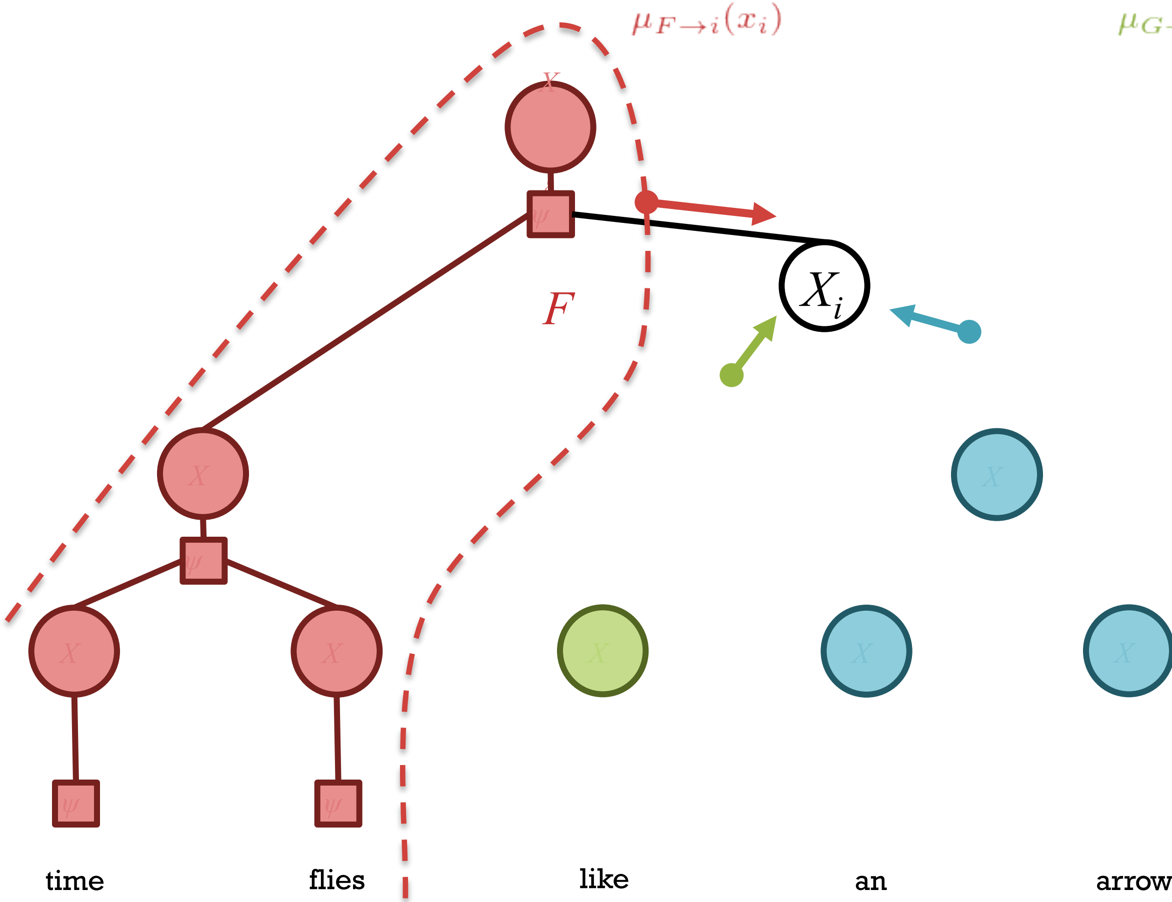
The marginal of X_i in that smaller model is the message sent to X_i from subgraph H

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

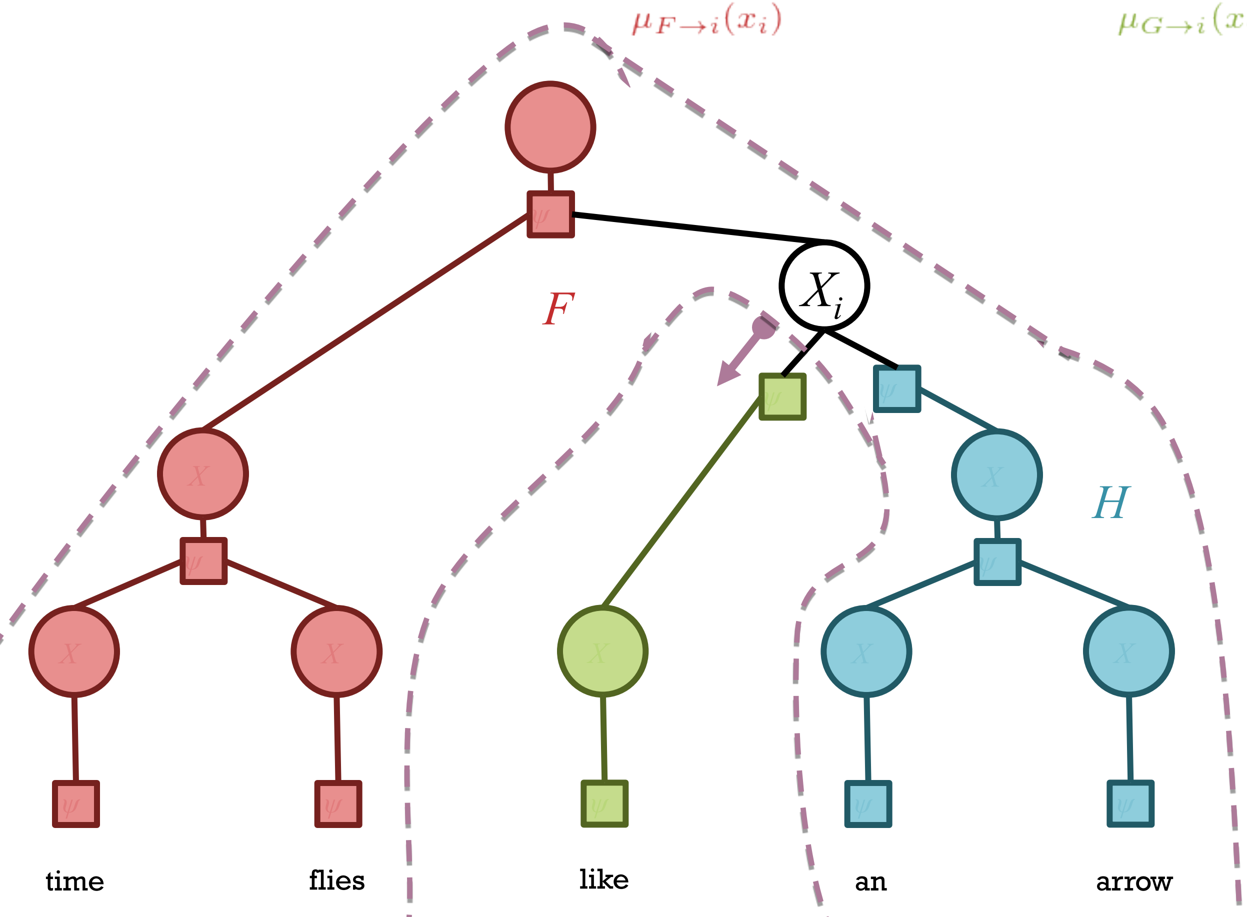
The marginal of X_i in that smaller model is the message sent to X_i from subgraph H

*Message **to** a variable*

Acyclic BP as Dynamic Programming

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

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



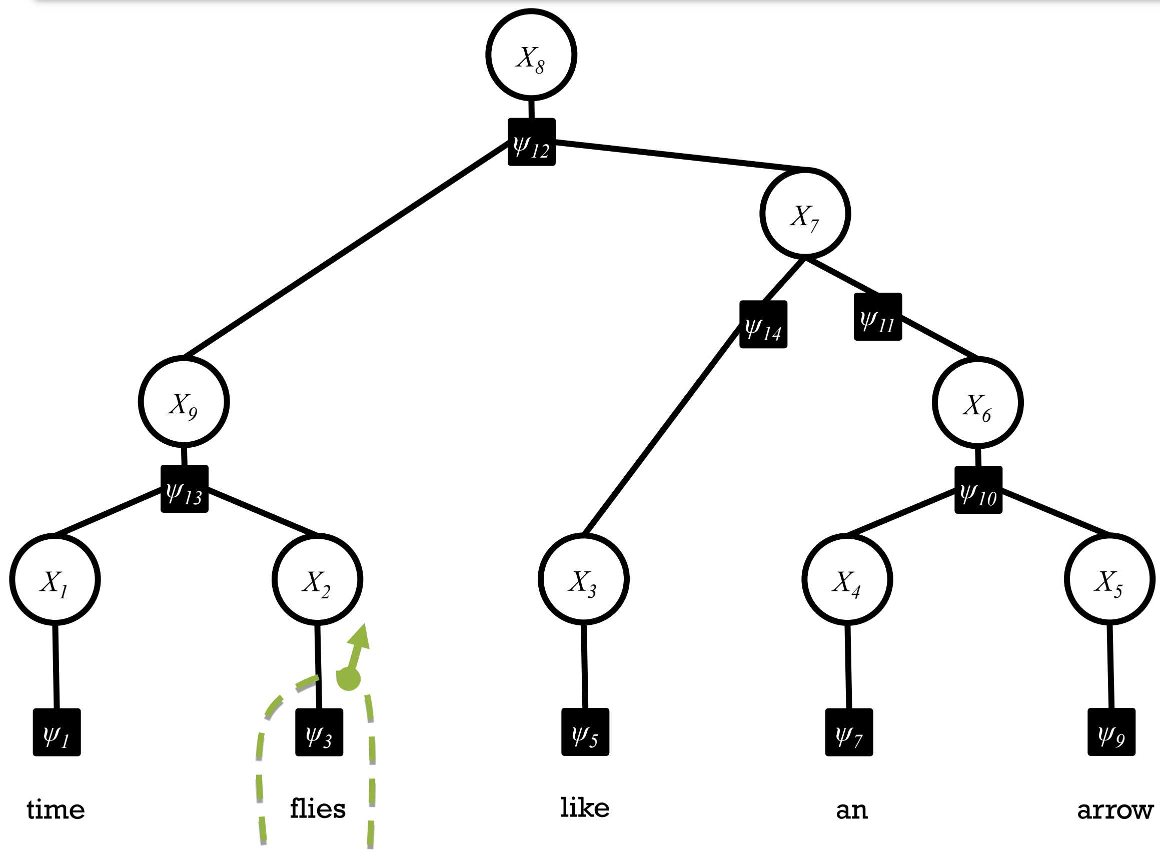
Subproblem:
Inference using just the factors in subgraph $F \cup H$

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

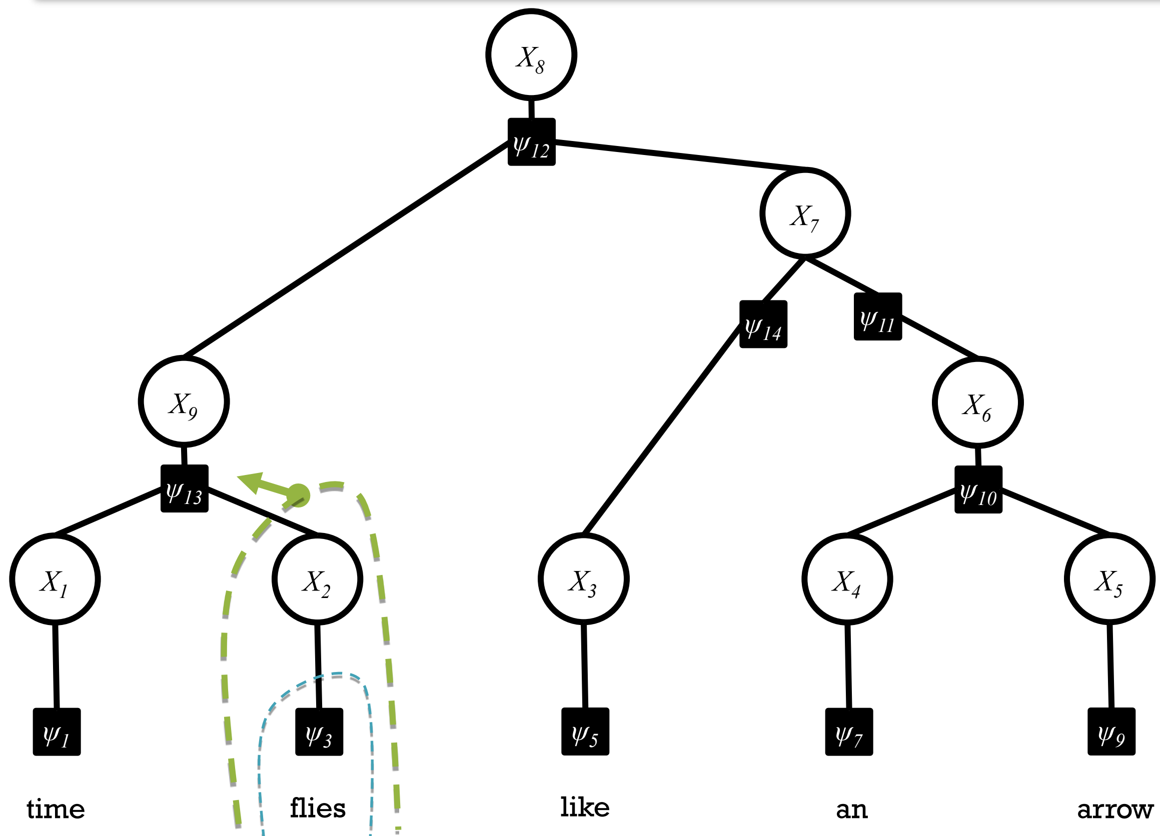
Acyclic BP as Dynamic Programming

- 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.



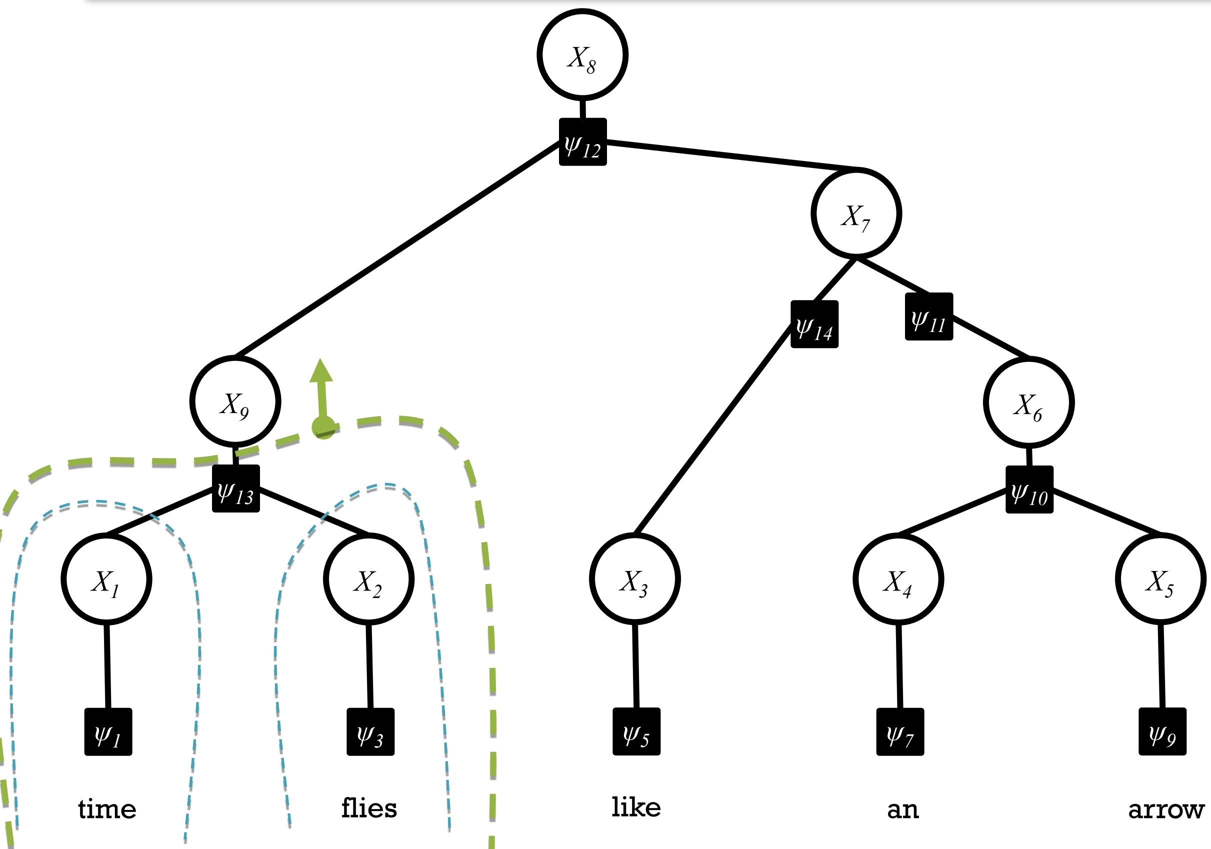
Acyclic BP as Dynamic Programming

- 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.



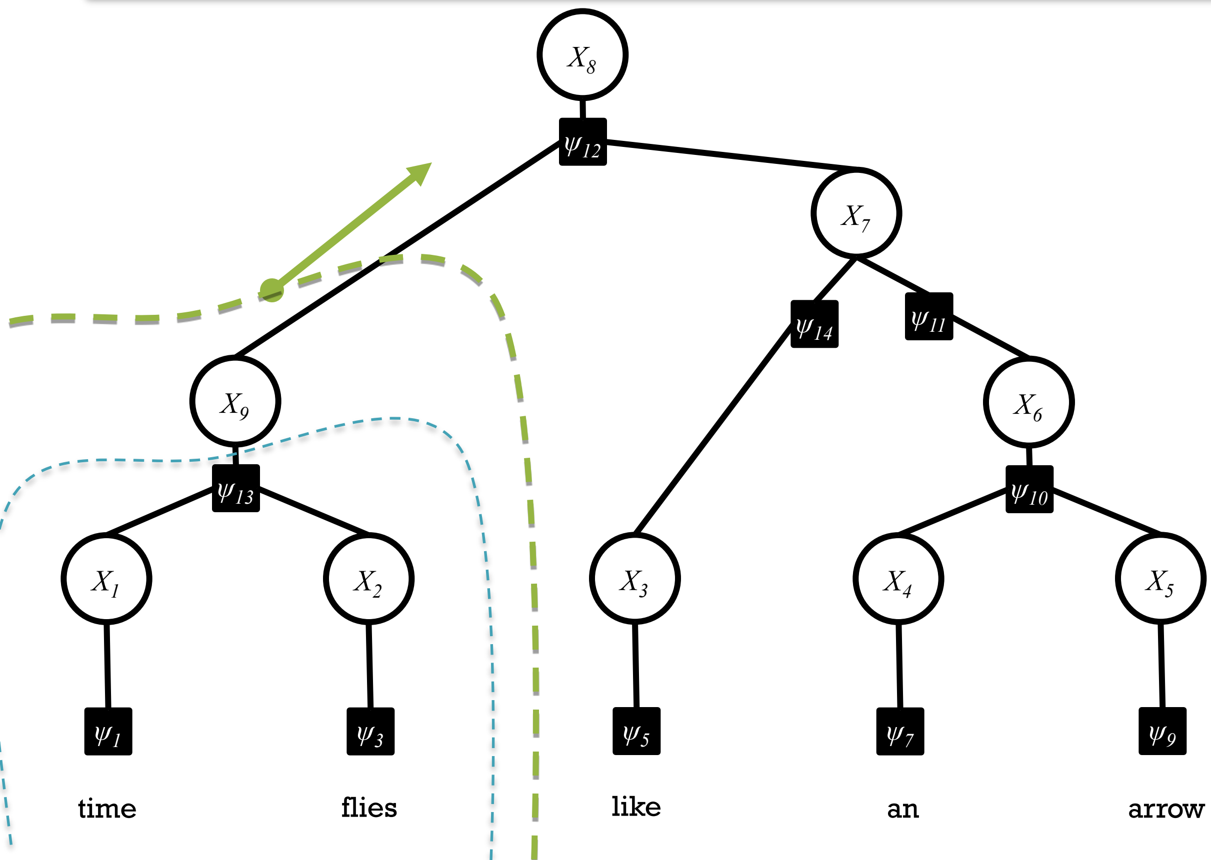
Acyclic BP as Dynamic Programming

- 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.



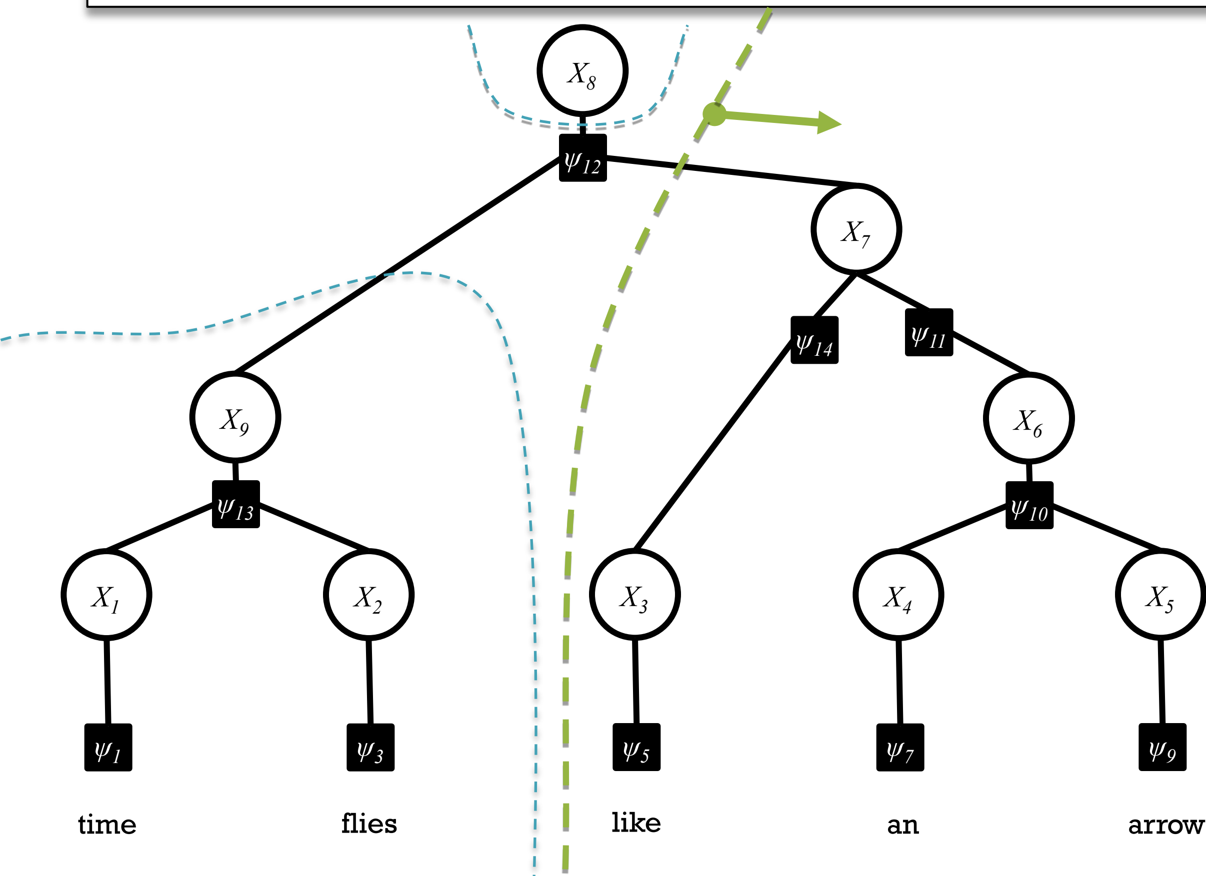
Acyclic BP as Dynamic Programming

- 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.



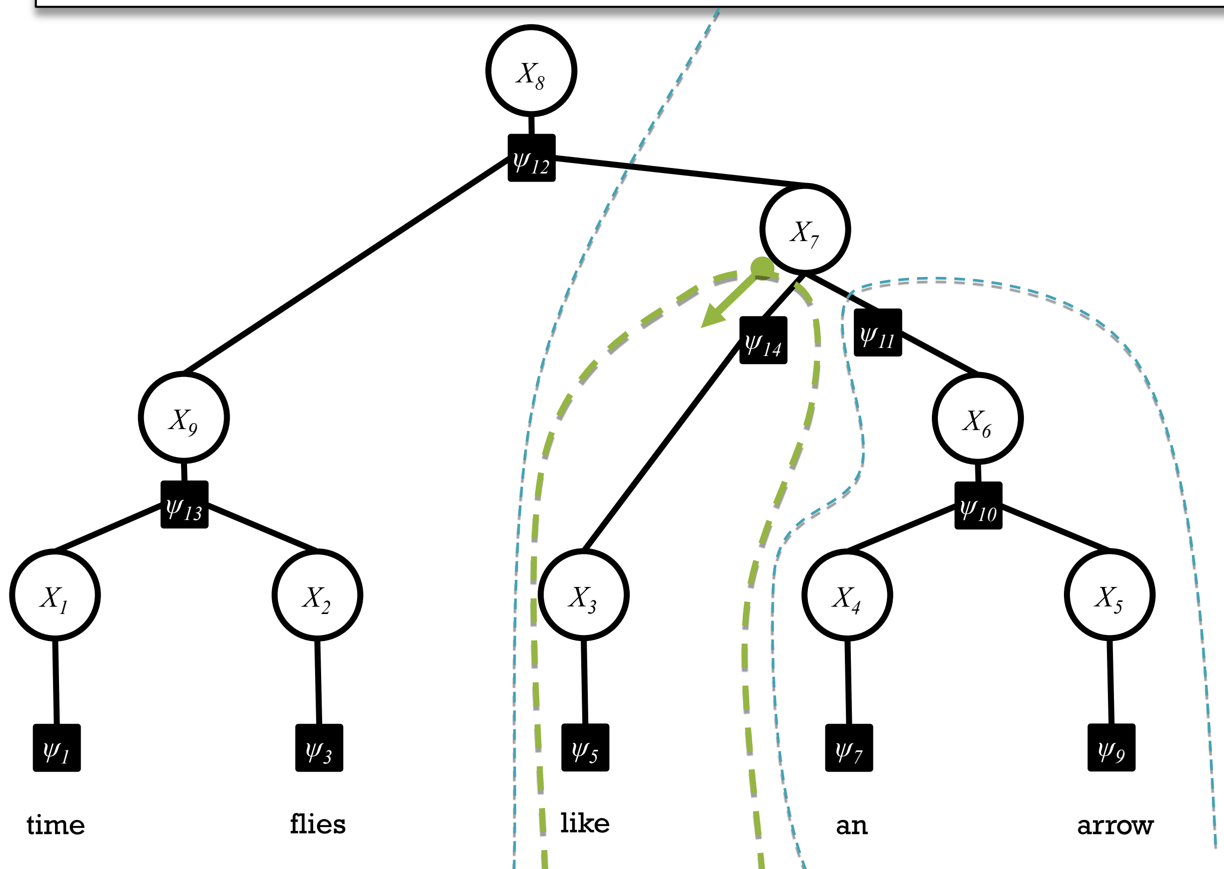
Acyclic BP as Dynamic Programming

- 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.



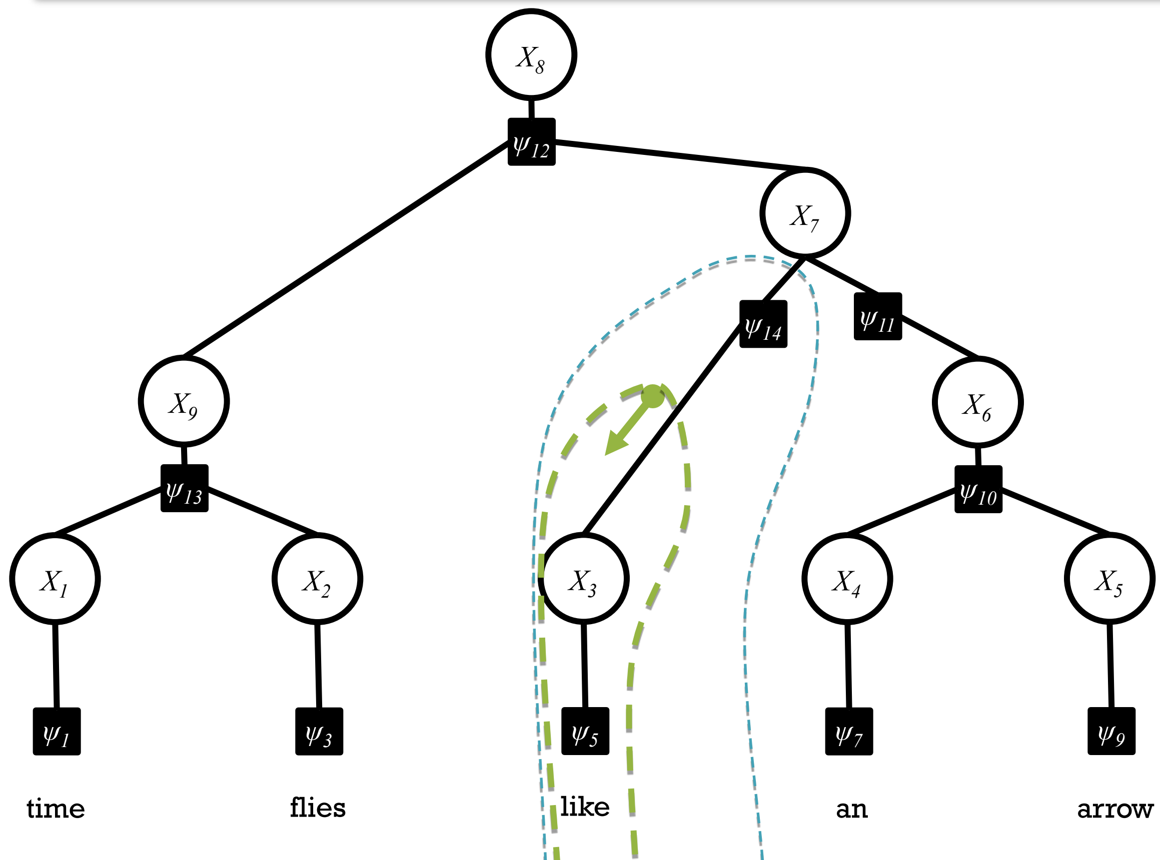
Acyclic BP as Dynamic Programming

- 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.



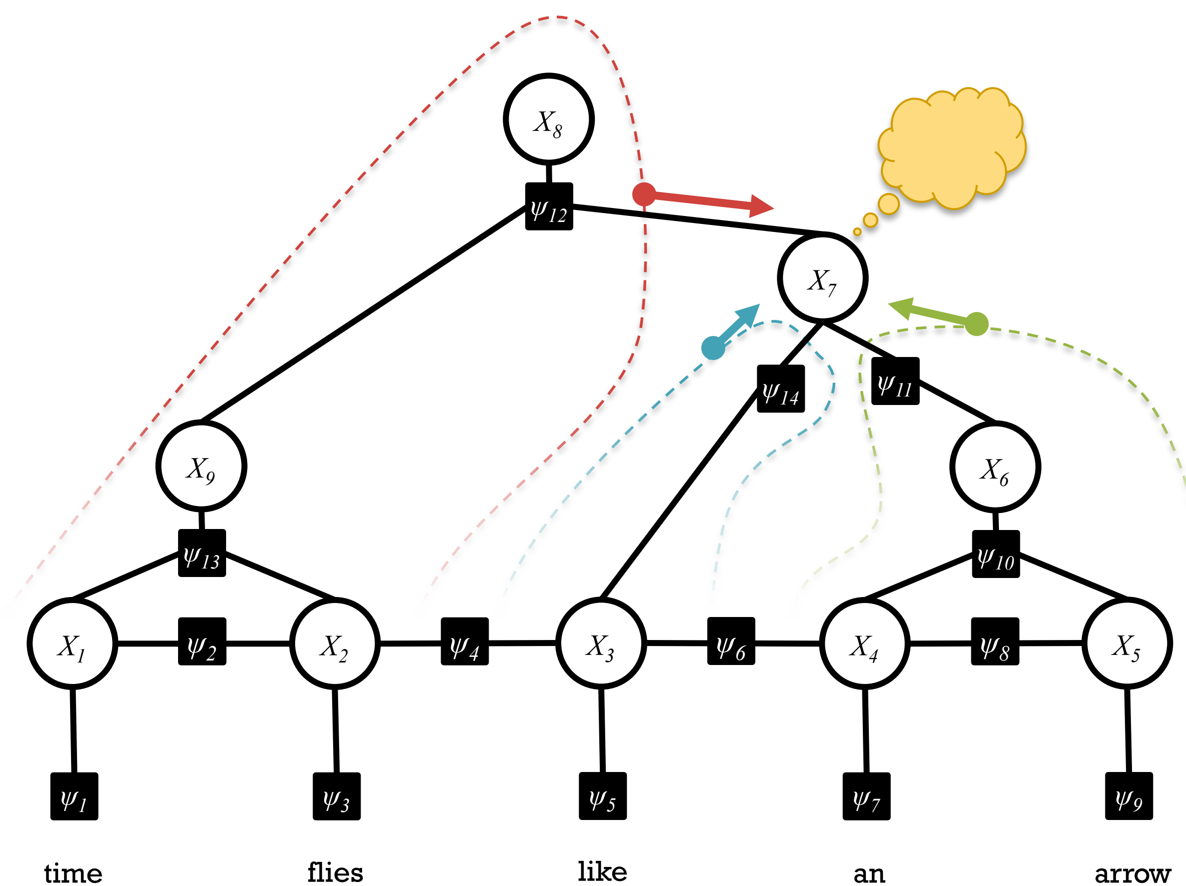
Acyclic BP as Dynamic Programming

- 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.



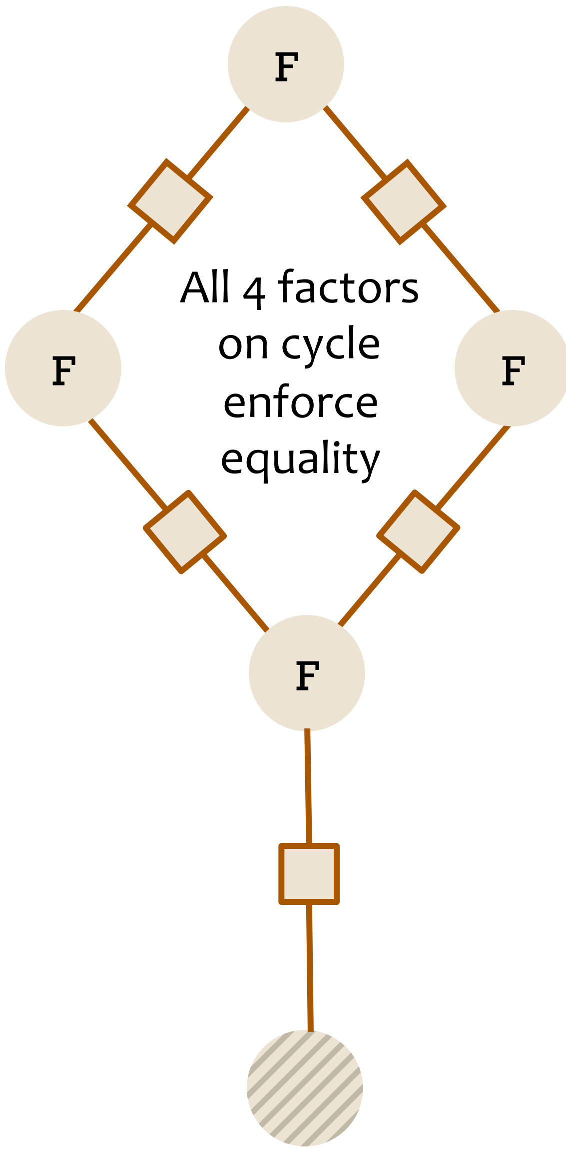
Loopy Belief Propagation

What if our graph has cycles?

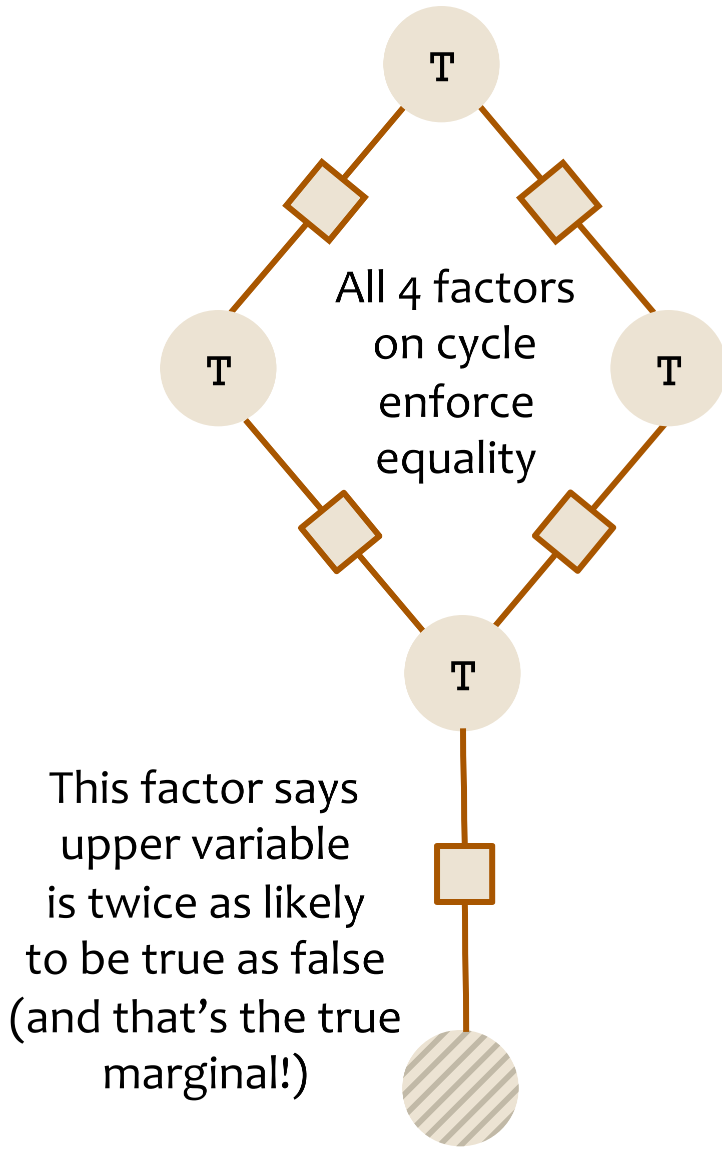


- Messages from different subgraphs are **no longer independent!**
  - Dynamic programming can't help. It's now #P-hard in general to compute the exact marginals.
- **But we can still run BP** -- it's a local algorithm so it doesn't "see the cycles."

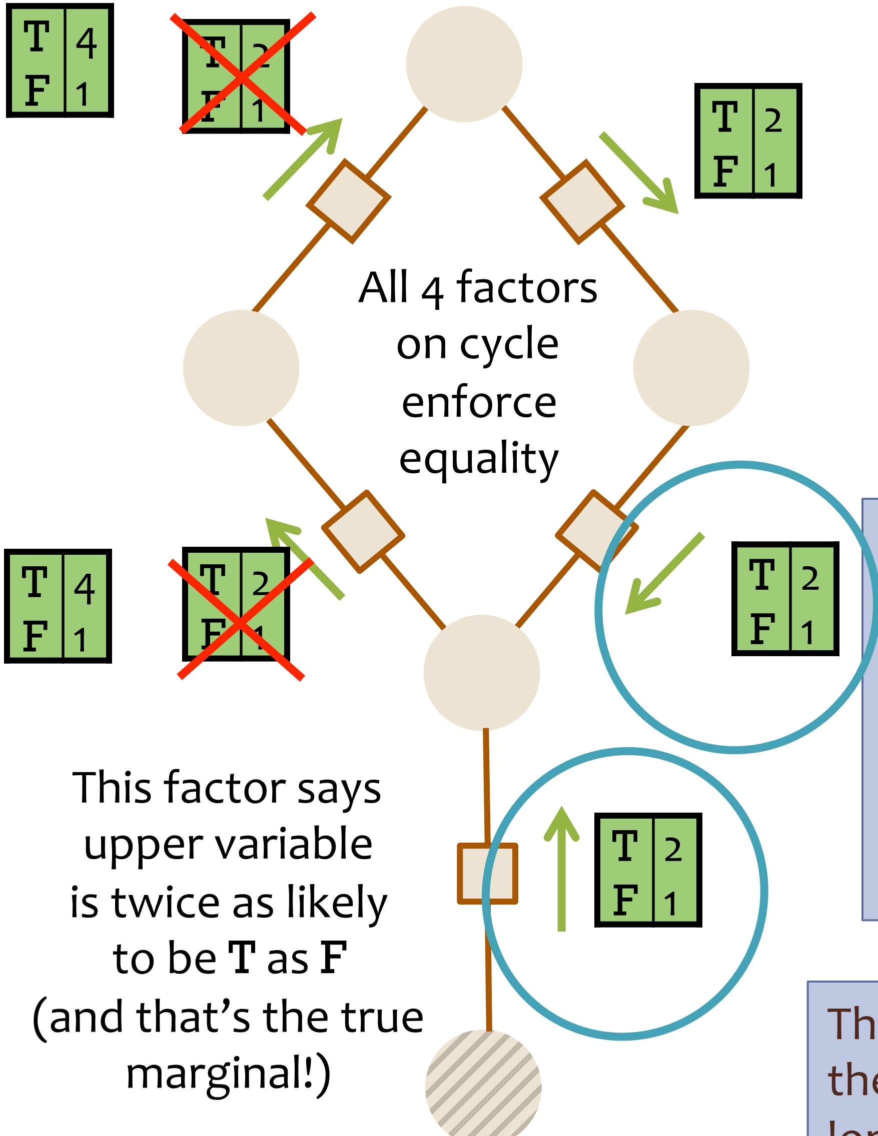
What can go wrong with loopy BP?



What can go wrong with loopy BP?



What can go wrong with loopy BP?



- Messages loop around and around …
- 2, 4, 8, 16, 32, ... More and more convinced that these variables are T!
- So beliefs converge to marginal distribution (1, 0) rather than (2/3, 1/3).

- BP incorrectly treats this message as separate evidence that the variable is T.
- Multiplies these two messages as if they were independent.
  - But they don't actually come from *independent* parts of the graph.
  - One influenced the other (via a cycle).

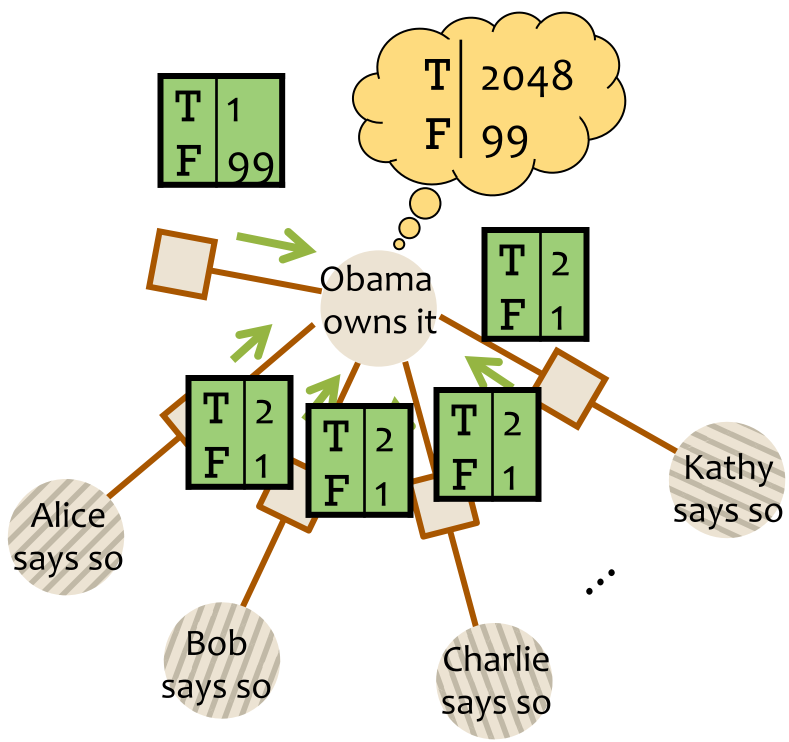
This is an extreme example. Often in practice, the cyclic influences are weak. (As cycles are long or include at least one weak correlation.)

What can go wrong with loopy BP?

Your prior doesn't think Obama owns it.

But everyone's saying he does. Under a Naïve Bayes model, you therefore believe it.

A rumor is circulating that Obama secretly owns an insurance company. (Obamacare is actually designed to maximize his profit.)



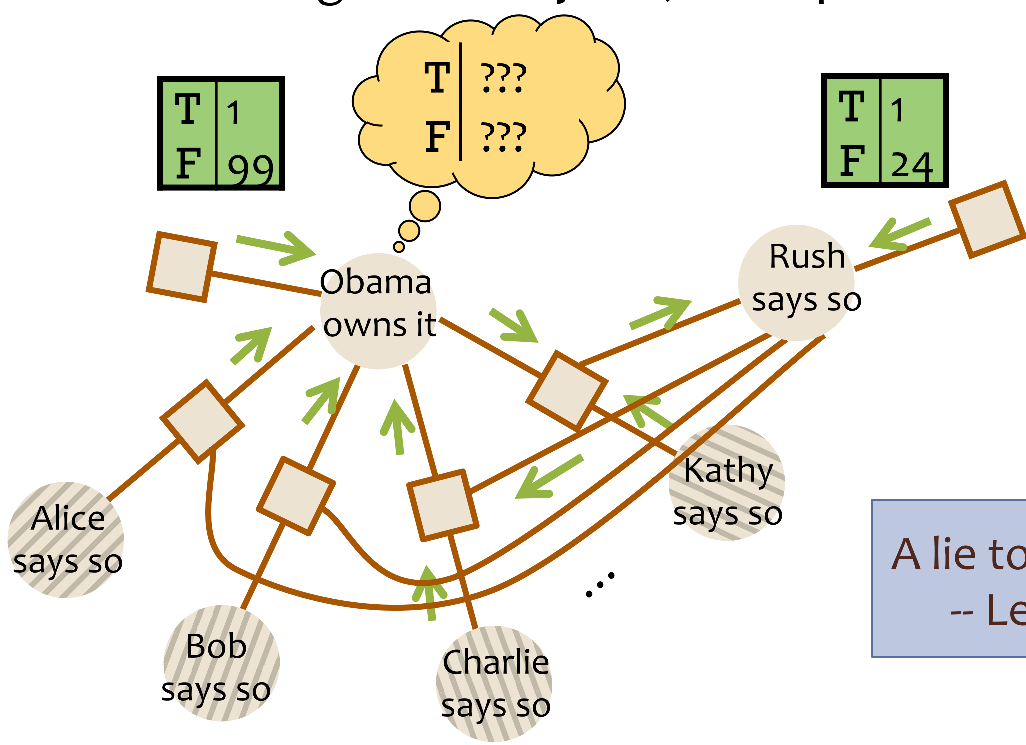
A lie told often enough becomes truth.
-- Lenin

What can go wrong with loopy BP?

Better model ... Rush can influence conversation.

- Now there are 2 ways to explain why everyone's repeating the story: it's true, or Rush said it was.
- The model favors one solution (probably Rush).
- Yet BP has 2 stable solutions. Each solution is self-reinforcing around cycles; no impetus to switch.

Actually 4 ways: but "both" has a low prior and "neither" has a low likelihood, so only 2 good ways.



If everyone blames Obama, then no one has to blame Rush.
But if no one blames Rush, then everyone has to continue to blame Obama (to explain the gossip).

~~A lie told often enough becomes truth.~~
~~-- Lenin~~

Loopy Belief Propagation Algorithm

- Run the BP update equations on a cyclic graph
  - Hope it "works" anyway (good approximation)
    - Though we multiply messages that aren't independent
    - No interpretation as dynamic programming
  - If largest element of a message gets very big or small,
    - Divide the message by a constant to prevent over/underflow
- Can update messages in any order
  - Stop when the normalized messages converge
- Compute beliefs from final messages
  - Return normalized beliefs as **approximate** marginals

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

e.g., Murphy, Weiss & Jordan (1999)

Loopy Belief Propagation

Input: a factor graph with cycles
Output: approximate marginals for each variable and factor

Algorithm:

1. Initialize the messages to the uniform distribution.

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

2. Send messages until convergence.
   Normalize them when they grow too large.

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

3. Compute the beliefs (unnormalized marginals).

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

4. Normalize beliefs and return the **approximate** marginals.

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

Section 2 Appendix

Tensor Notation for BP

Tensor Notation for BP

In section 2, BP was introduced with a notation which defined messages and beliefs as functions.

This Appendix includes an alternate (and very concise) notation for the Belief Propagation algorithm using tensors.

Tensor Notation

- Tensor multiplication:

$$(A \otimes B)\,(W = w, X = x, Y = y) = A(W = w, X = x)B(X = x, Y = y)$$

- Tensor marginalization:

$$\left(\bigoplus^{Y} A\right)(Y = y) = \sum_{w}\sum_{x} A(W = w, X = x, Y = y)$$

Tensor Notation

A rank-r tensor is…

A real function with r keyword arguments = Axis-labeled array with arbitrary indices = Database with column headers

Tensor multiplication: (vector outer product)

$$
(A \otimes B)\,(X = x, Y = y) \\
= A(X = x) B(Y = y)
$$

$$
X \begin{array}{c|c|} 1 & 3 \\ 2 & 5 \end{array} \otimes Y \begin{array}{c|c|} \text{red} & 4 \\ \text{blue} & 6 \end{array}
$$

		Y	
		red	blue
X	1	12	18
	2	20	30

X	value
1	3
2	5

$\otimes$

Y	value
red	4
blue	6

X	Y	value
1	red	12
2	red	20
1	blue	18
2	blue	30

Tensor Notation

A rank-r tensor is…

A real function with r keyword arguments = Axis-labeled array with arbitrary indices = Database with column headers

Tensor multiplication: (vector pointwise product)

$$(A \otimes B)\,(X = x)$$
$$= A(X = x)B(X = x)$$

$$X \begin{array}{c|c|} \cline{2-2} \mathtt{a} & 3 \\ \cline{2-2} \mathtt{b} & 5 \\ \cline{2-2} \end{array} \otimes X \begin{array}{c|c|} \cline{2-2} \mathtt{a} & 4 \\ \cline{2-2} \mathtt{b} & 6 \\ \cline{2-2} \end{array}$$

$$\downarrow$$

$$X \begin{array}{c|c|} \cline{2-2} \mathtt{a} & 12 \\ \cline{2-2} \mathtt{b} & 30 \\ \cline{2-2} \end{array}$$

X	value
a	3
b	5

$\otimes$

X	value
a	4
b	6

$\downarrow$

X	value
a	12
b	30

Tensor Notation

A rank-r tensor is…

A real function with r keyword arguments = Axis-labeled array with arbitrary indices = Database with column headers

Tensor multiplication: (matrix-vector product)

$$(A \otimes B)(X = x, Y = y) = A(X = x, Y = y)B(X = x)$$

$$\begin{array}{cc|cc} & & \multicolumn{2}{c}{Y} \\ & & \text{red} & \text{blue} \\ X & 1 & 3 & 4 \\ & 2 & 5 & 6 \end{array} \otimes \begin{array}{cc|c} X & 1 & 7 \\ & 2 & 8 \end{array}$$

$$\downarrow$$

$$\begin{array}{cc|cc} & & \multicolumn{2}{c}{Y} \\ & & \text{red} & \text{blue} \\ X & 1 & 21 & 28 \\ & 2 & 40 & 48 \end{array}$$

X	Y	value
1	red	3
2	red	5
1	blue	4
2	blue	6

$\otimes$

X	value
1	7
2	8

$\downarrow$

X	Y	value
1	red	21
2	red	40
1	blue	28
2	blue	48

Tensor Notation

A rank-r tensor is…

A real function with r keyword arguments = Axis-labeled array with arbitrary indices = Database with column headers

Tensor marginalization:

$$\left(\bigoplus^{Y} A\right)(Y=y) = \sum_{x} A(X=x, Y=y)$$

X \ Y	red	blue
1	3	4
2	5	6

$\downarrow \bigoplus^{X}$

Y	red	blue
	8	10

X	Y	value
1	red	3
2	red	5
1	blue	4
2	blue	6

$\downarrow \bigoplus^{X}$

Y	value
red	8
blue	10

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.

$$\boxed{\mu_{i\to\alpha} = 1} \quad \boxed{\mu_{\alpha\to i} = 1}$$

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

$$\boxed{\mu_{i\to\alpha} = \bigotimes_{\beta\in\mathcal{N}(i)\backslash\alpha} \mu_{\beta\to i}} \quad \boxed{\mu_{\alpha\to i} = \bigoplus^{i} \psi_\alpha \otimes \bigotimes_{j\in\mathcal{N}(\alpha)\backslash i} \mu_{j\to\alpha}}$$

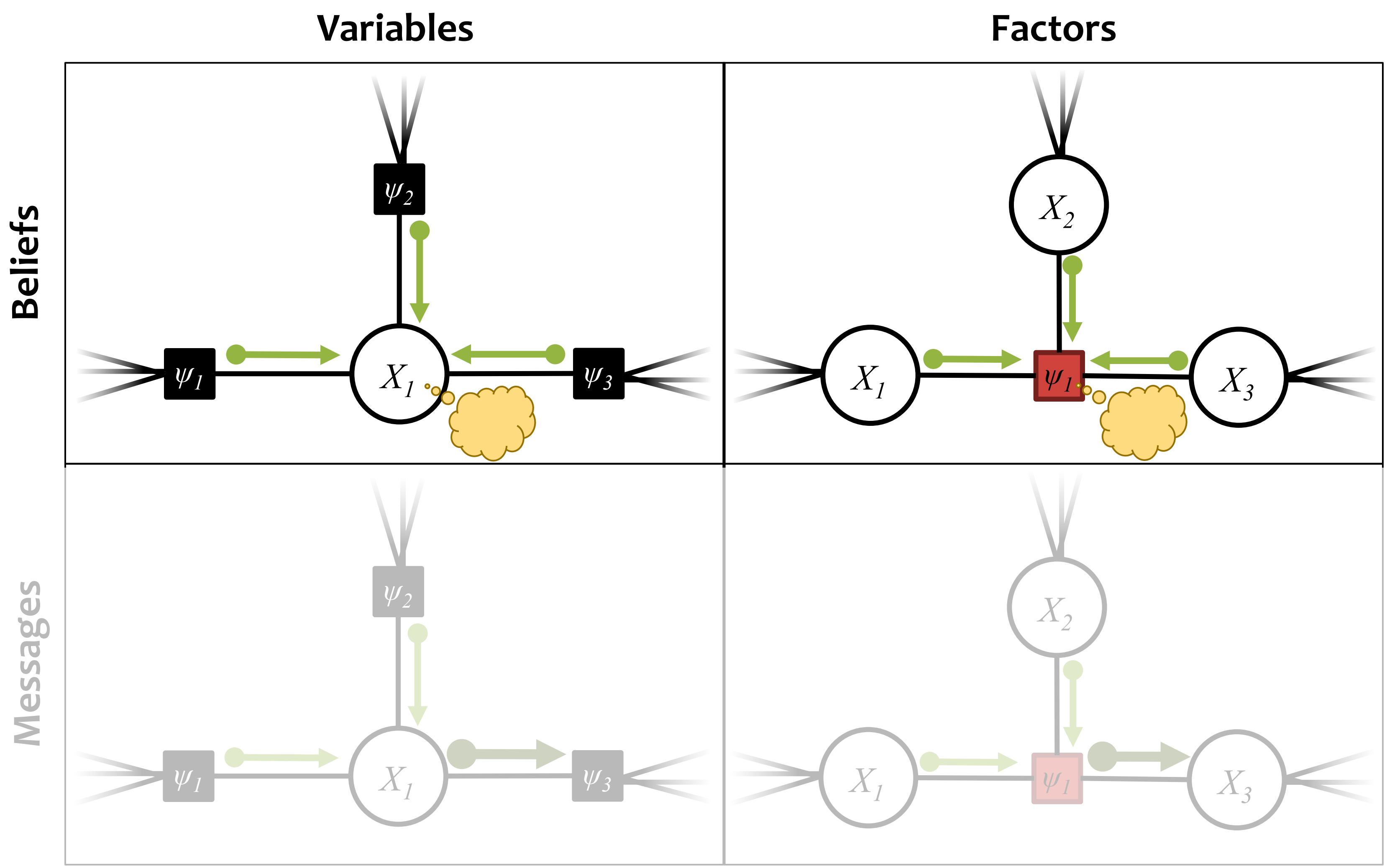
4. Compute the beliefs (unnormalized marginals).

$$\boxed{b_i = \bigotimes_{\alpha\in\mathcal{N}(i)} \mu_{\alpha\to i}} \quad \boxed{b_\alpha = \psi_\alpha \otimes \bigotimes_{i\in\mathcal{N}(\alpha)} \mu_{i\to\alpha}}$$

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

$$\boxed{p_i(x_i) \propto b_i(x_i)} \quad \boxed{p_\alpha(\boldsymbol{x_\alpha}) \propto b_\alpha(\boldsymbol{x_\alpha})}$$

Sum-Product Belief Propagation



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

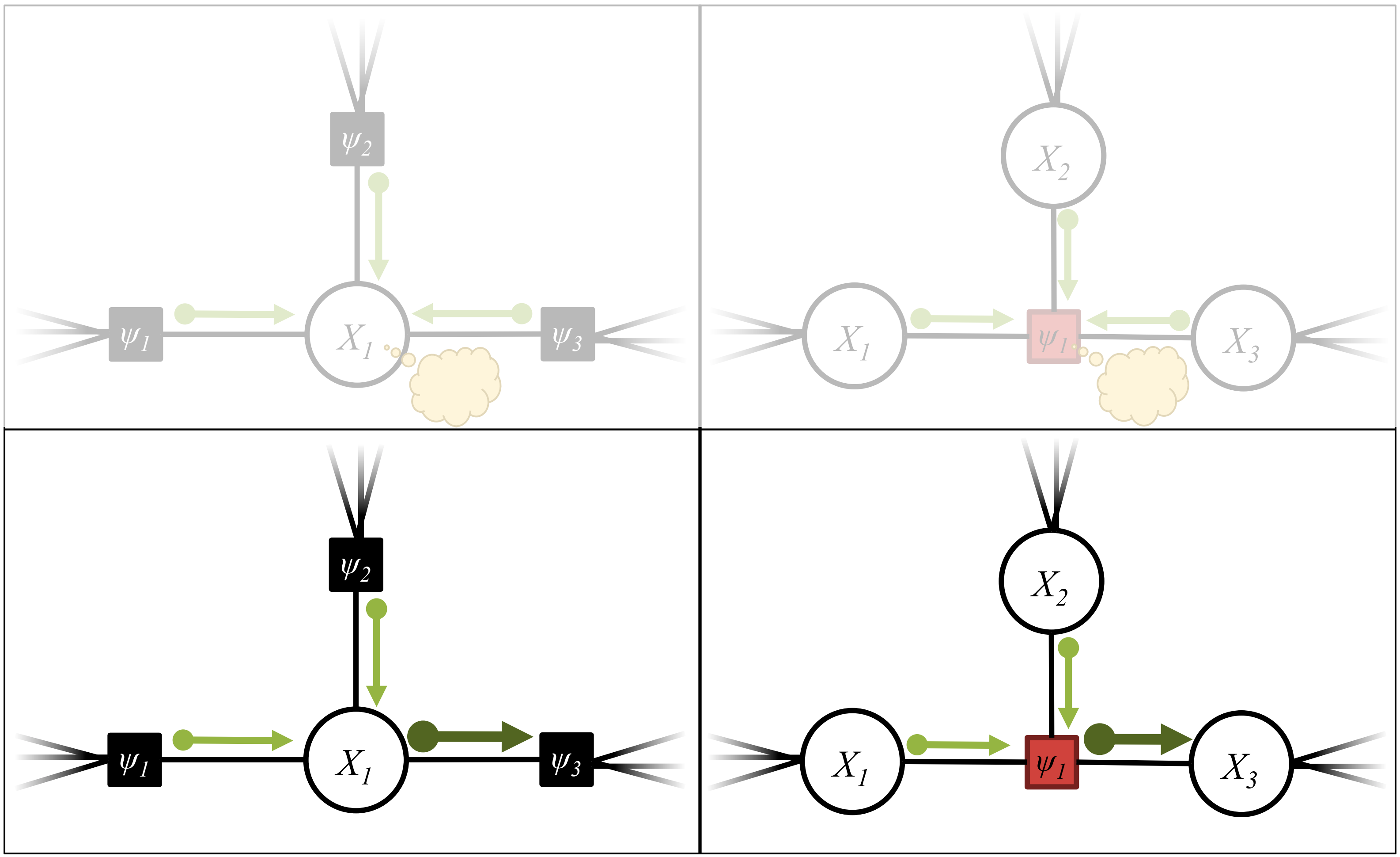
$$b_\alpha = \psi_\alpha \otimes \bigotimes_{i \in \mathcal{N}(\alpha)} \mu_{i \to \alpha}$$

Sum-Product Belief Propagation

Variables | Factors

Beliefs

Messages

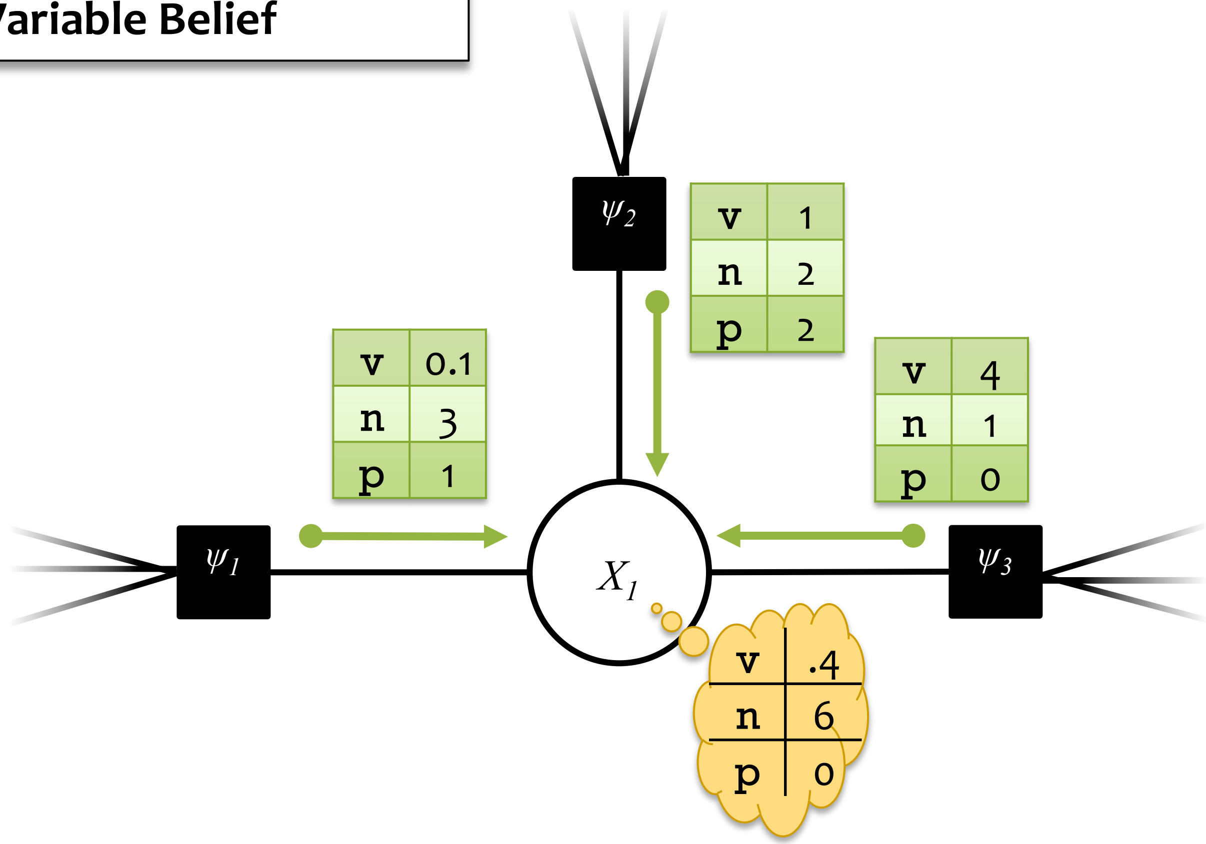


$$\mu_{i\to\alpha} = \bigotimes_{\beta\in\mathcal{N}(i)\setminus\alpha} \mu_{\beta\to i}$$

$$\mu_{\alpha\to i} = \bigoplus^{i} \psi_\alpha \otimes \bigotimes_{j\in\mathcal{N}(\alpha)\setminus i} \mu_{j\to\alpha}$$

Sum-Product Belief Propagation

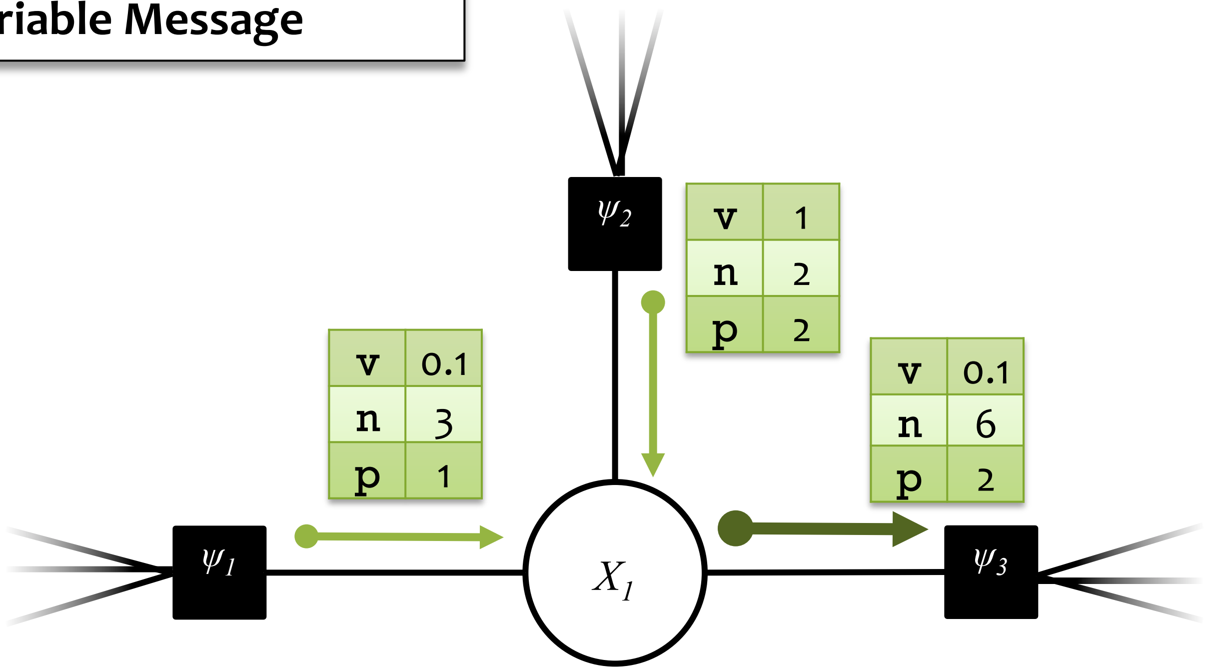
Variable Belief



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

Sum-Product Belief Propagation

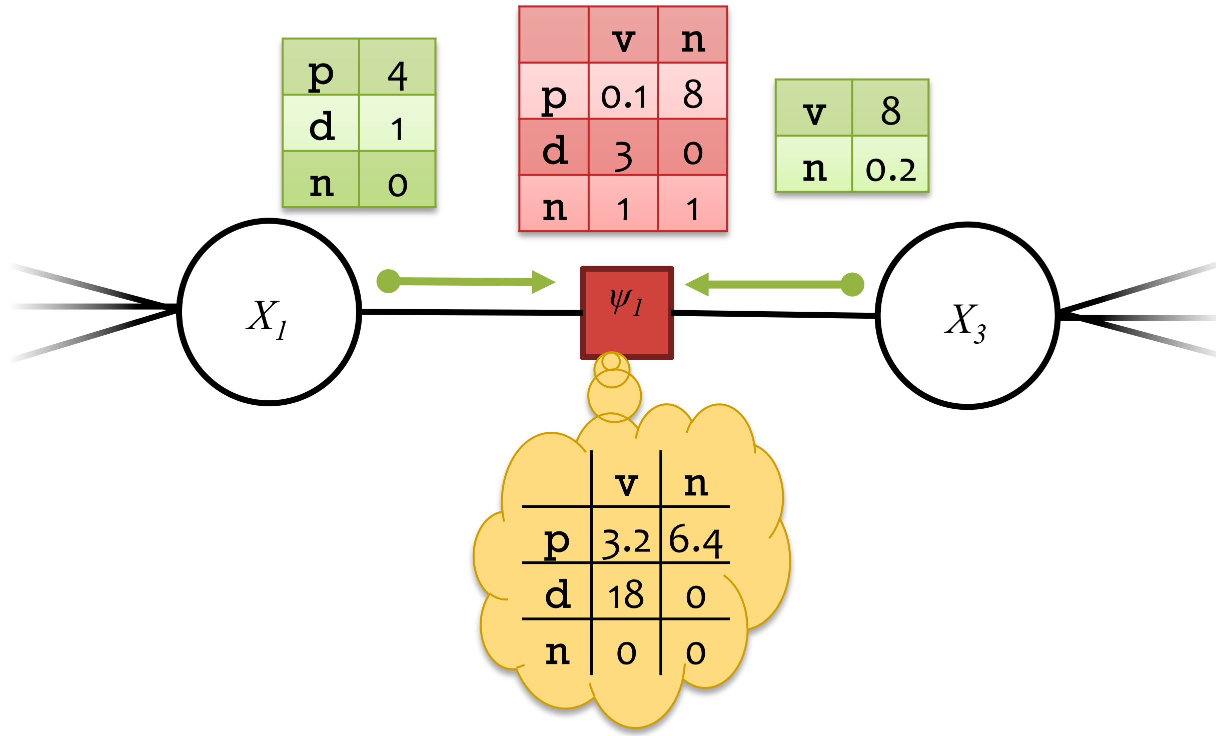
Variable Message



$$\mu_{i \to \alpha} = \bigotimes_{\beta \in \mathcal{N}(i) \setminus \alpha} \mu_{\beta \to i}$$

Sum-Product Belief Propagation

Factor Belief



$$b_\alpha = \psi_\alpha \otimes \bigotimes_{i \in \mathcal{N}(\alpha)} \mu_{i \to \alpha}$$

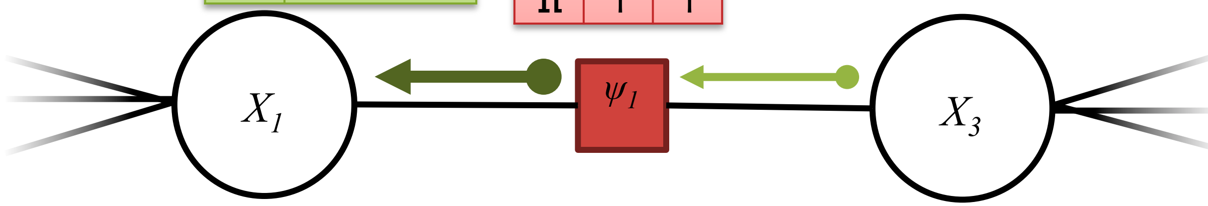
Sum-Product Belief Propagation

Factor Message

p	0.8 + 0.16
d	24 + 0
n	8 + 0.2

	v	n
p	0.1	8
d	3	0
n	1	1

v	8
n	0.2



$$\mu_{\alpha\to i} = \bigoplus^{i} \psi_\alpha \otimes \bigotimes_{j\in\mathcal{N}(\alpha)\setminus i} \mu_{j\to\alpha}$$

Loopy Belief Propagation

Input: a factor graph with cycles
Output: approximate marginals for each variable and factor

Algorithm:

1. Initialize the messages to the uniform distribution.

$$\boxed{\mu_{i \to \alpha} = 1} \quad \boxed{\mu_{\alpha \to i} = 1}$$

2. Send messages until convergence.
   Normalize them when they grow too large.

$$\boxed{\mu_{i \to \alpha} = \bigotimes_{\beta \in \mathcal{N}(i) \setminus \alpha} \mu_{\beta \to i}} \quad \boxed{\mu_{\alpha \to i} = \bigoplus^{i} \psi_\alpha \otimes \bigotimes_{j \in \mathcal{N}(\alpha) \setminus i} \mu_{j \to \alpha}}$$

3. Compute the beliefs (unnormalized marginals).

$$\boxed{b_i = \bigotimes_{\alpha \in \mathcal{N}(i)} \mu_{\alpha \to i}} \quad \boxed{b_\alpha = \psi_\alpha \otimes \bigotimes_{i \in \mathcal{N}(\alpha)} \mu_{i \to \alpha}}$$

4. Normalize beliefs and return the **approximate** marginals.

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

Section 3: Belief Propagation Q&A

Methods like BP and in what sense they work

Outline

- Do you want to push past the simple NLP models (logistic regression, PCFG, etc.) that we've all been using for 20 years?
- Then this tutorial is extremely practical for you!
  1. **Models:** Factor graphs can express interactions among linguistic structures.
  2. **Algorithm:** BP estimates the global effect of these interactions on each variable, using local computations.
  3. **Intuitions:** What's going on here? Can we trust BP's estimates?
  4. **Fancier Models:** Hide a whole grammar and dynamic programming algorithm within a single factor. BP coordinates multiple factors.
  5. **Tweaked Algorithm:** Finish in fewer steps and make the steps faster.
  6. **Learning:** Tune the parameters. Approximately improve the true predictions -- or truly improve the approximate predictions.
  7. **Software:** Build the model you want!

Outline

- Do you want to push past the simple NLP models (logistic regression, PCFG, etc.) that we've all been using for 20 years?
- Then this tutorial is extremely practical for you!
  1. **Models:** Factor graphs can express interactions among linguistic structures.
  2. **Algorithm:** BP estimates the global effect of these interactions on each variable, using local computations.
  3. **Intuitions:** What's going on here? Can we trust BP's estimates?
  4. **Fancier Models:** Hide a whole grammar and dynamic programming algorithm within a single factor. BP coordinates multiple factors.
  5. **Tweaked Algorithm:** Finish in fewer steps and make the steps faster.
  6. **Learning:** Tune the parameters. Approximately improve the true predictions -- or truly improve the approximate predictions.
  7. **Software:** Build the model you want!

Q&A

Q: Forward-backward is to the Viterbi algorithm as sum-product BP is to ____________ ?

A: max-product BP

Max-product Belief Propagation

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

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

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:

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

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

- **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

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

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

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

4. Take resulting beliefs to power T

Q&A

Q: This feels like **Arc Consistency**…
Any relation?

A: Yes, BP is doing (with probabilities) what people were doing in AI long before.

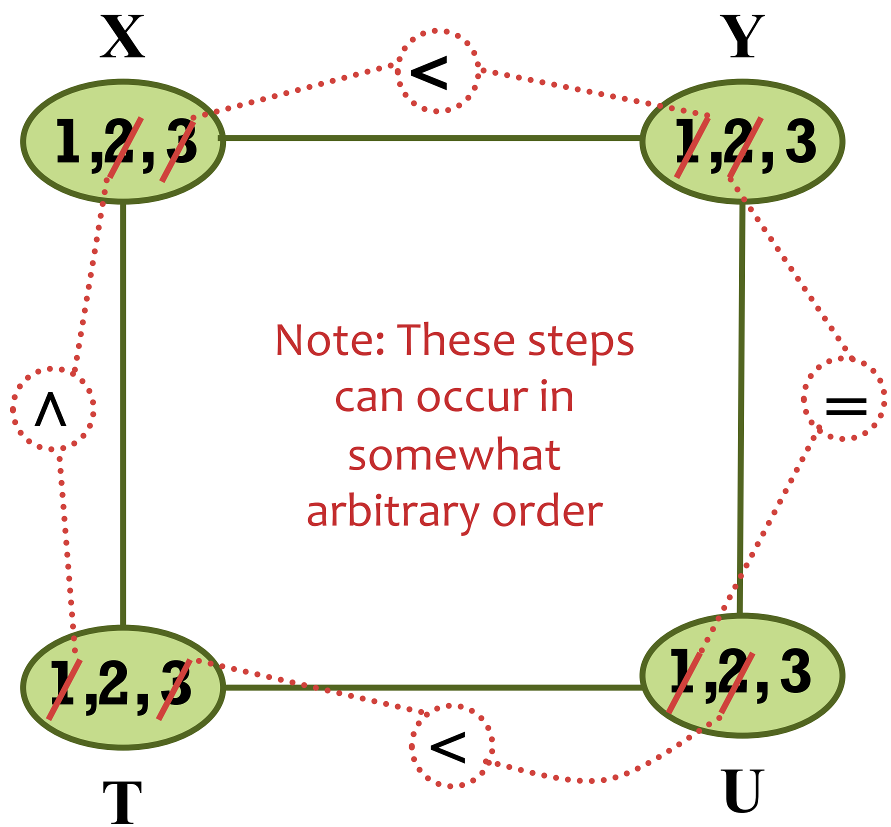
From Arc Consistency to BP

Goal: Find a satisfying assignment

Algorithm: Arc Consistency

1. Pick a constraint
2. Reduce domains to satisfy the constraint
3. Repeat until convergence

$$X, Y, U, T \in \{1, 2, 3\}$$
$$X < Y$$
$$Y = U$$
$$T < U$$
$$X < T$$



Propagation completely solved the problem!

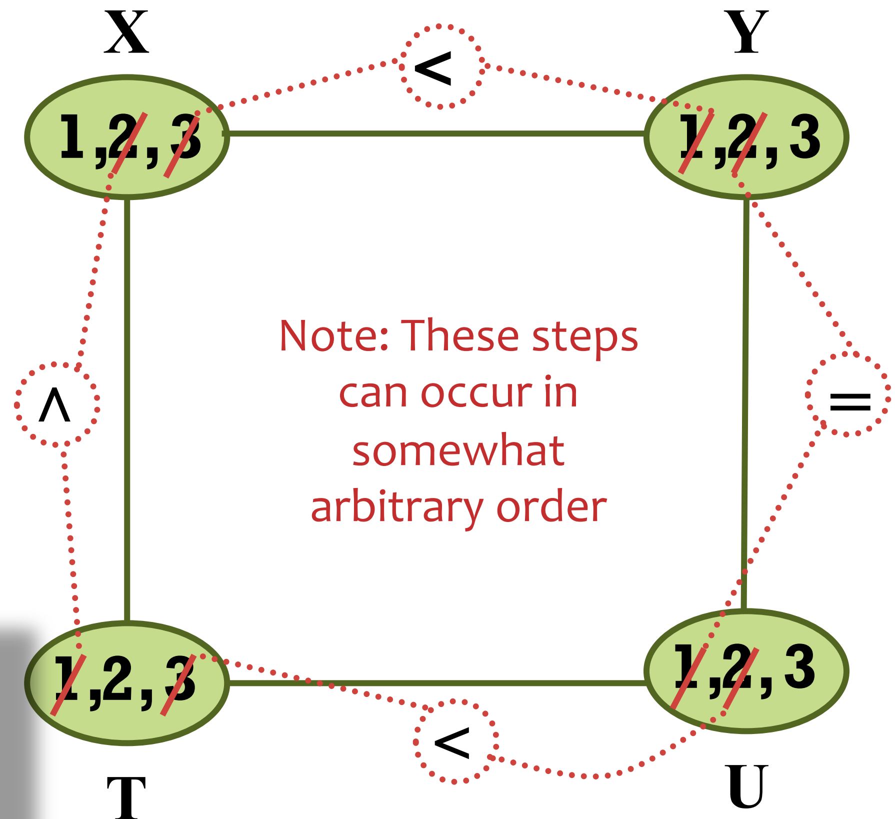
Slide thanks to Rina Dechter (modified)

From Arc Consistency to BP

Goal: Find a satisfying assignment
Algorithm: Arc Consistency

1. Pick a constraint
2. Reduce domains to satisfy the constraint
3. Repeat until convergence

X, Y, U, T ∈ {1, 2, 3}
X < Y
Y = U
T < U
X < T



Arc Consistency is a special case of Belief Propagation.

Propagation completely solved the problem!

Slide thanks to Rina Dechter (modified)

From Arc Consistency to BP

Solve the same problem with BP

- Constraints become "hard" factors with only 1's or 0's
- Send messages until convergence

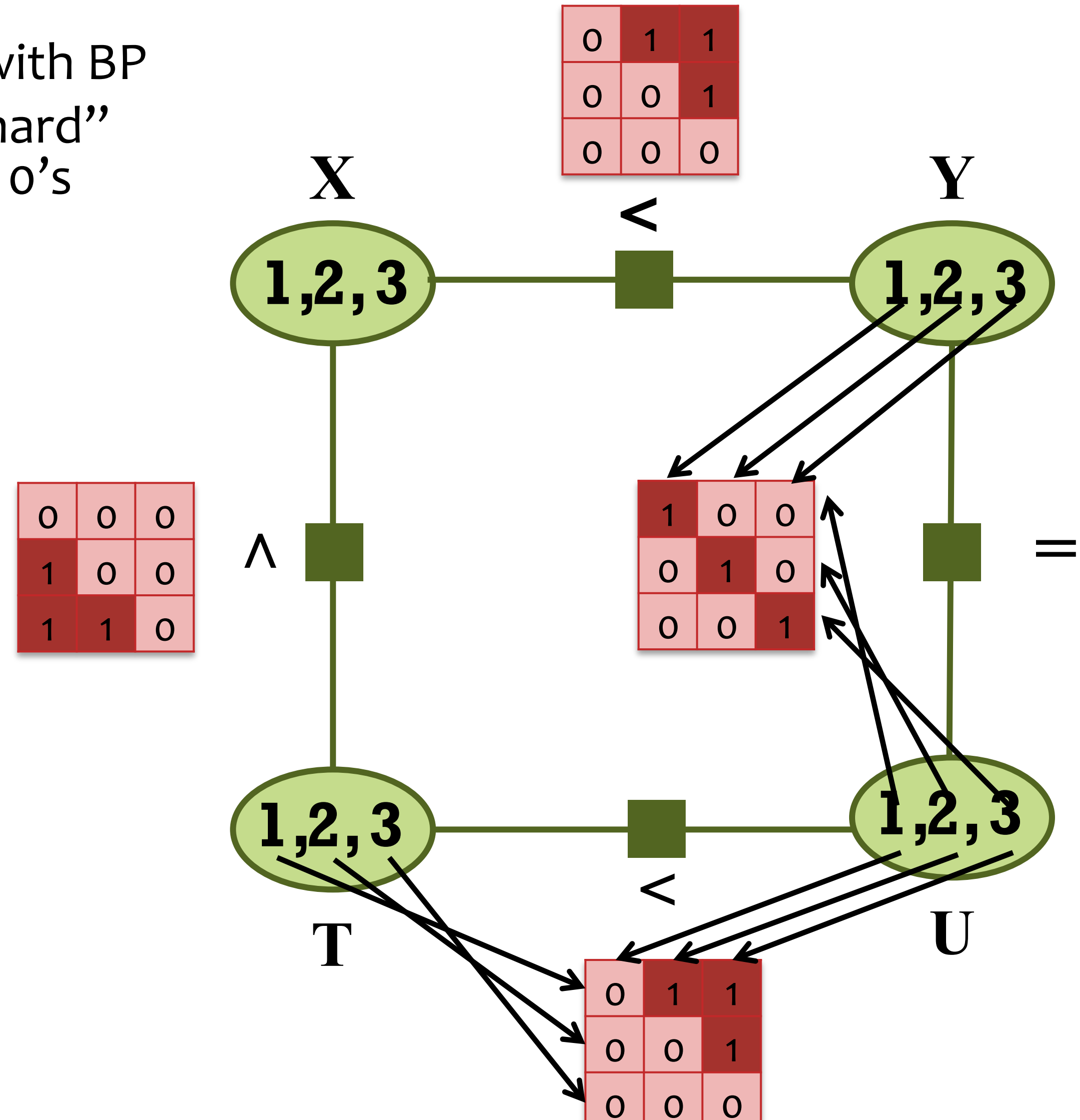
X, Y, U, T ∈ {1, 2, 3}

X < Y

Y = U

T < U

X < T



Slide thanks to Rina Dechter (modified)

From Arc Consistency to BP

Solve the same problem with BP
- Constraints become "hard" factors with only 1's or 0's
- Send messages until convergence

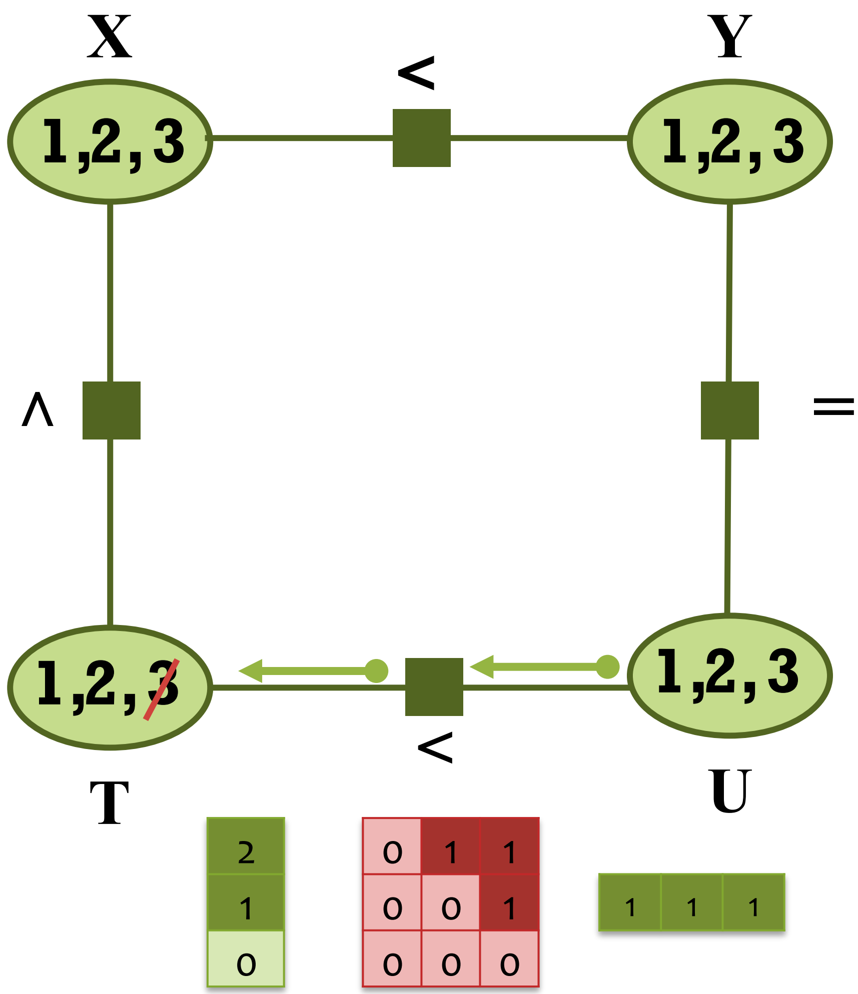
X, Y, U, T ∈ {1, 2, 3}

X < Y

Y = U

T < U

X < T



Slide thanks to Rina Dechter (modified)

From Arc Consistency to BP

Solve the same problem with BP
- Constraints become "hard" factors with only 1's or 0's
- Send messages until convergence

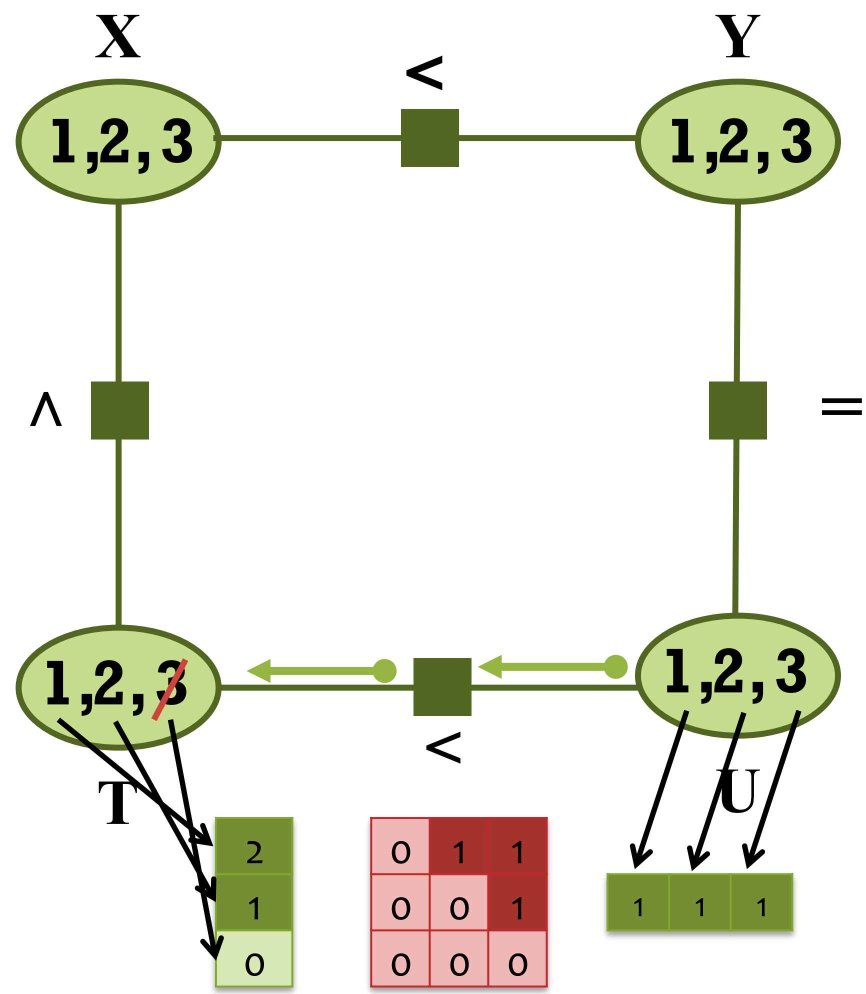
X, Y, U, T ∈ {1, 2, 3}

X < Y

Y = U

T < U

X < T



Slide thanks to Rina Dechter (modified)

From Arc Consistency to BP

Solve the same problem with BP
- Constraints become "hard" factors with only 1's or 0's
- Send messages until convergence

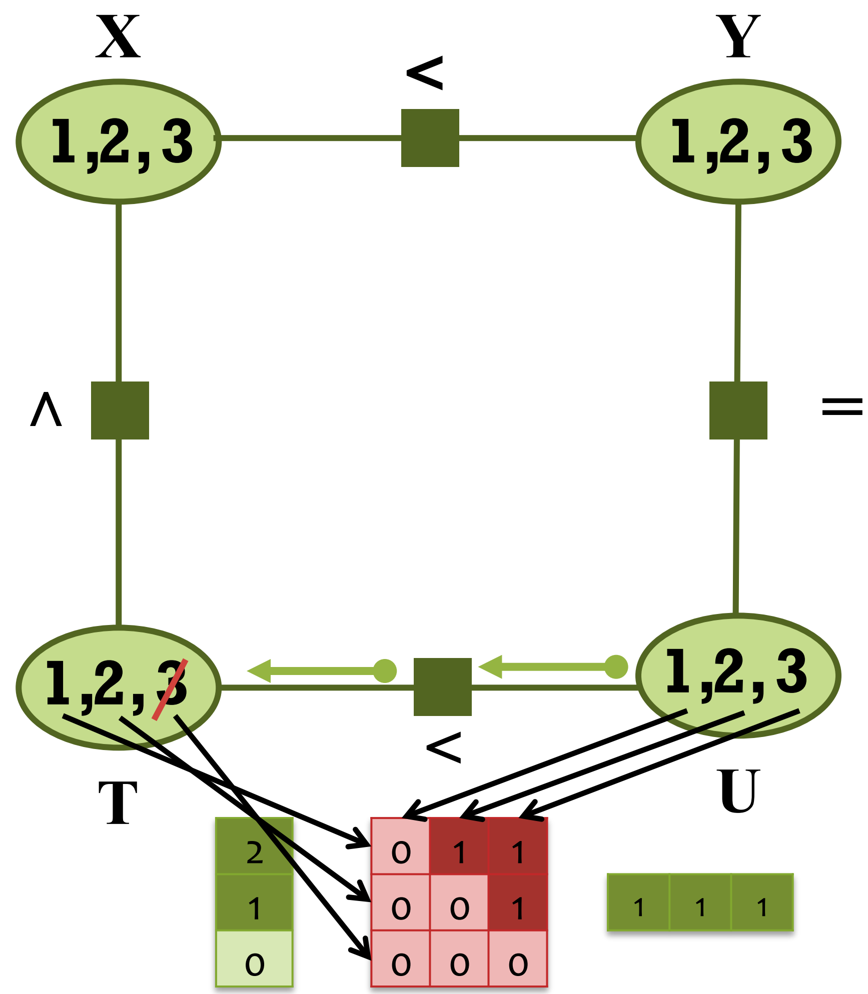
X, Y, U, T ∈ {1, 2, 3}

X < Y

Y = U

T < U

X < T



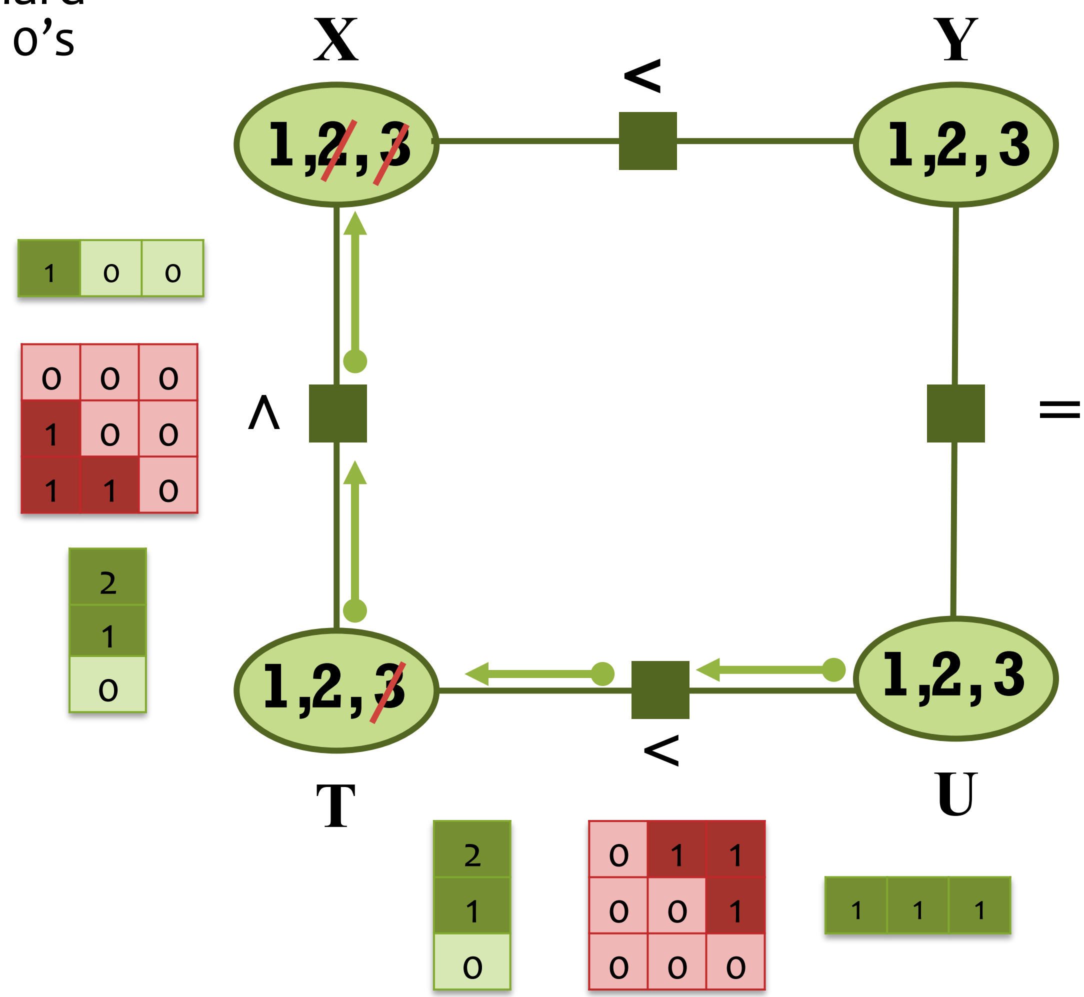
Slide thanks to Rina Dechter (modified)

From Arc Consistency to BP

Solve the same problem with BP

- Constraints become “hard” factors with only 1’s or 0’s
- Send messages until convergence

$$X, Y, U, T \in \{1, 2, 3\}$$
$$X < Y$$
$$Y = U$$
$$T < U$$
$$X < T$$



Slide thanks to Rina Dechter (modified)

From Arc Consistency to BP

Solve the same problem with BP

- Constraints become "hard" factors with only 1's or 0's
- Send messages until convergence

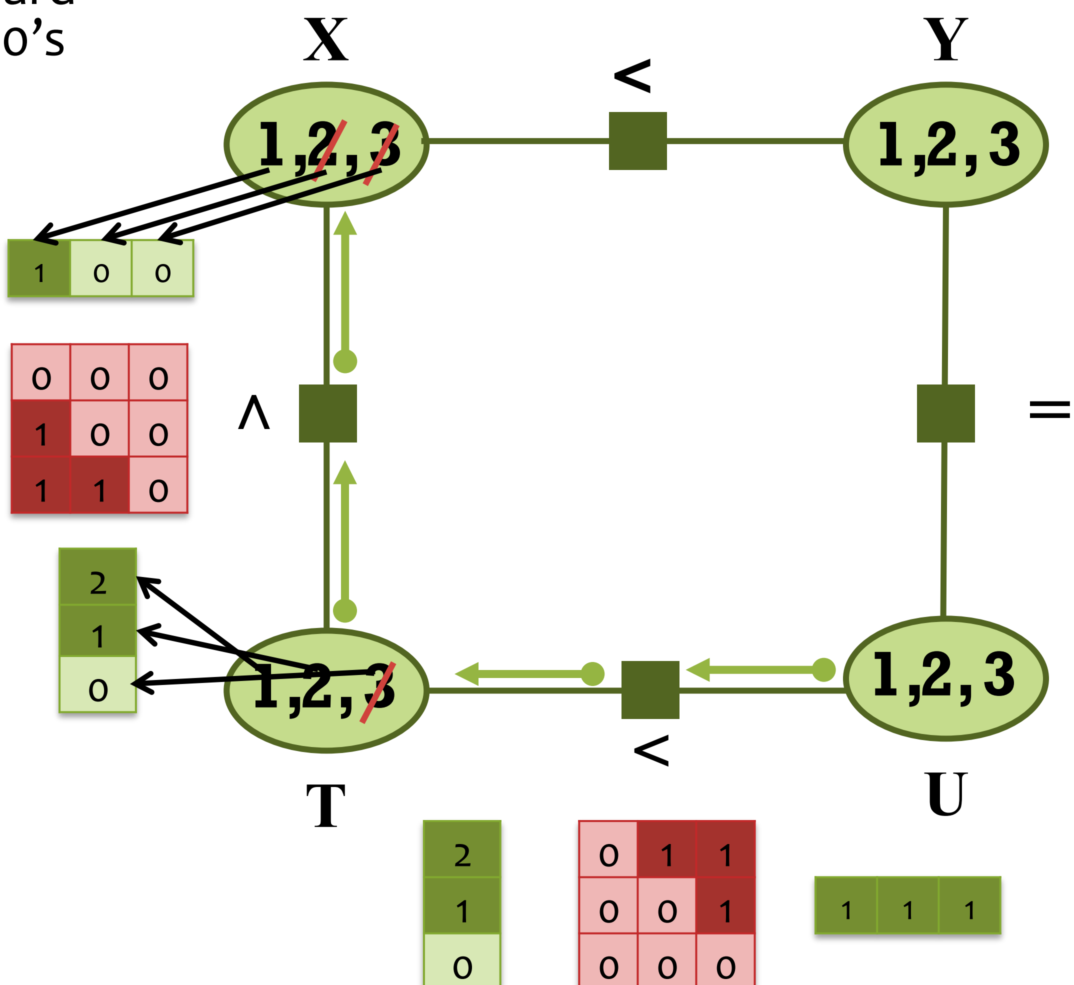
X, Y, U, T ∈ {1, 2, 3}

X < Y

Y = U

T < U

X < T



Slide thanks to Rina Dechter (modified)

From Arc Consistency to BP

Solve the same problem with BP
- Constraints become "hard" factors with only 1's or 0's
- Send messages until convergence

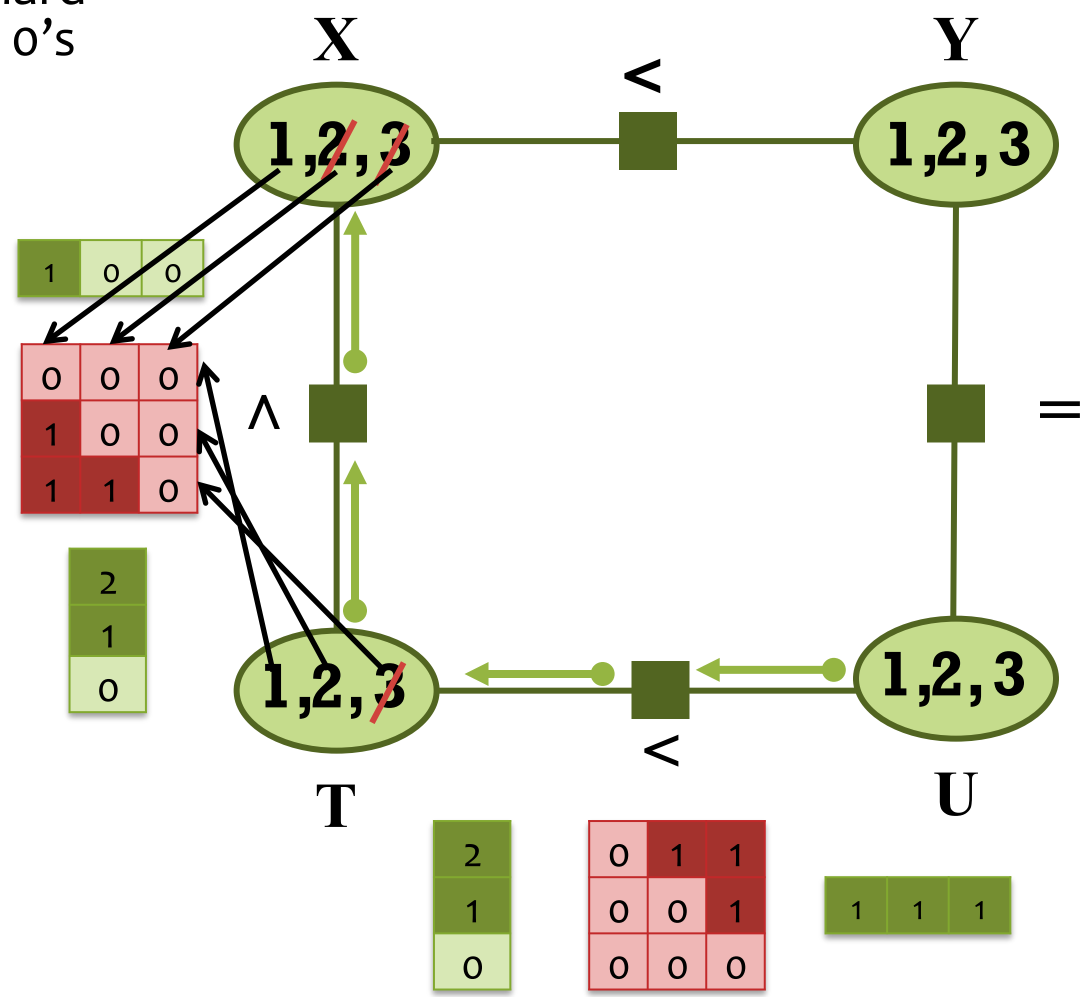
$X, Y, U, T \in \{1, 2, 3\}$

$X < Y$

$Y = U$

$T < U$

$X < T$



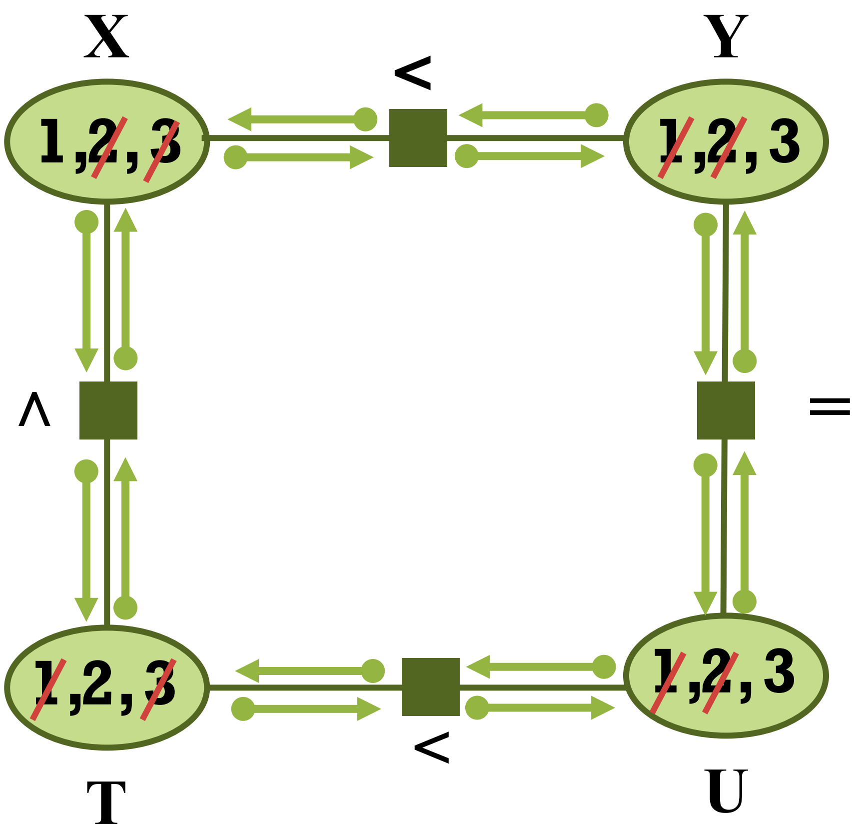
Slide thanks to Rina Dechter (modified)

From Arc Consistency to BP

Solve the same problem with BP

- Constraints become "hard" factors with only 1's or 0's
- Send messages until convergence

$$X, Y, U, T \in \{1, 2, 3\}$$
$$X < Y$$
$$Y = U$$
$$T < U$$
$$X < T$$



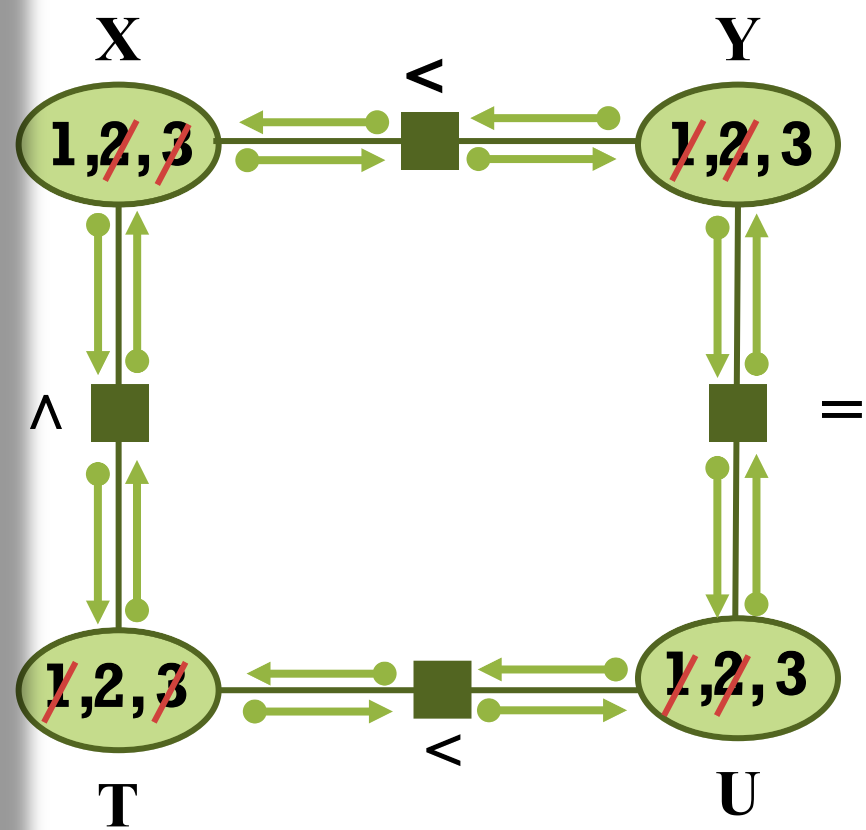
Loopy BP will converge to the equivalent solution!

Slide thanks to Rina Dechter (modified)

From Arc Consistency to BP

Takeaways:

- Arc Consistency is a special case of Belief Propagation.
- Arc Consistency will only rule out impossible values.
- BP rules out those same values (belief = 0).



Loopy BP will converge to the equivalent solution!

Slide thanks to Rina Dechter (modified)

Q&A

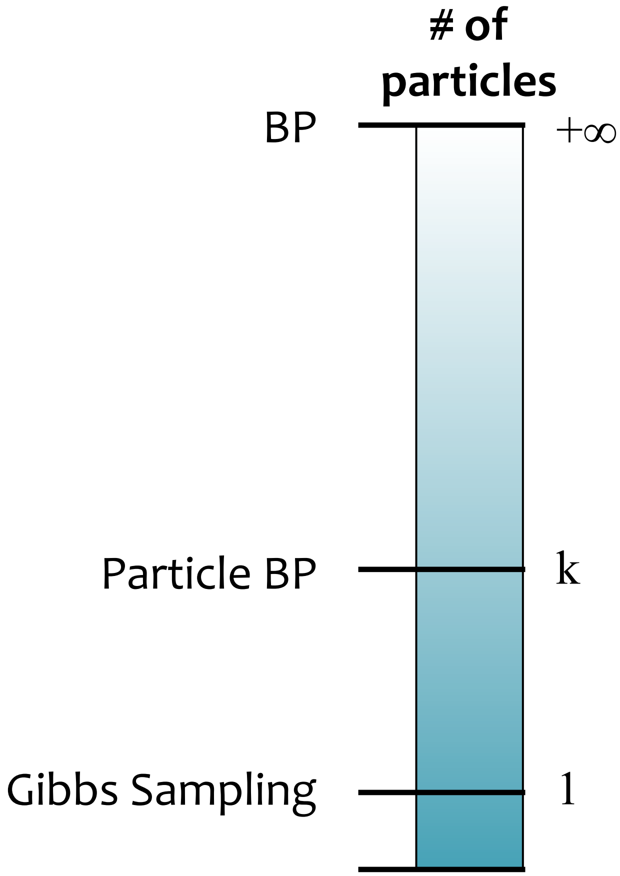
Q: Is BP totally divorced from sampling?

A: Gibbs Sampling is also a kind of message passing algorithm.

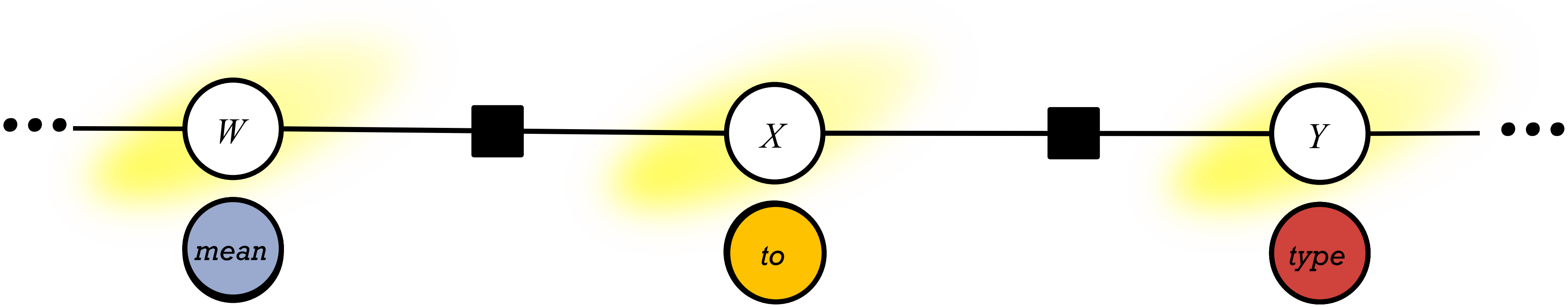
From Gibbs Sampling to Particle BP to BP

Message Representation:

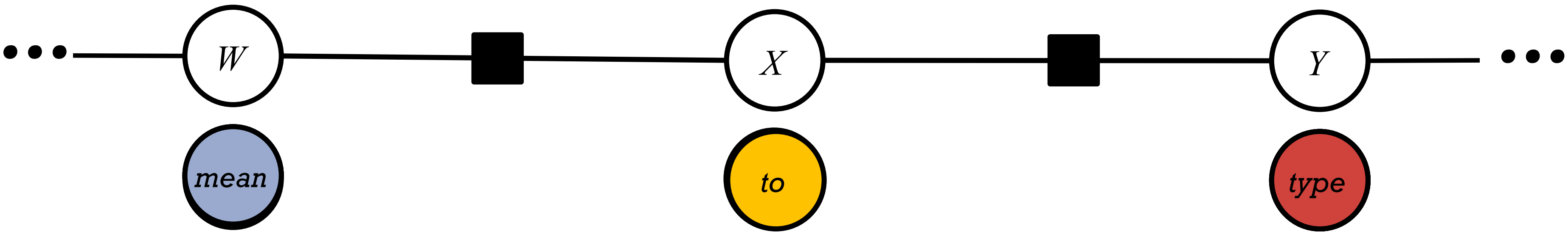
A. Belief Propagation: **full distribution**
B. Gibbs sampling: **single particle**
C. Particle BP: **multiple particles**



From Gibbs Sampling to Particle BP to BP



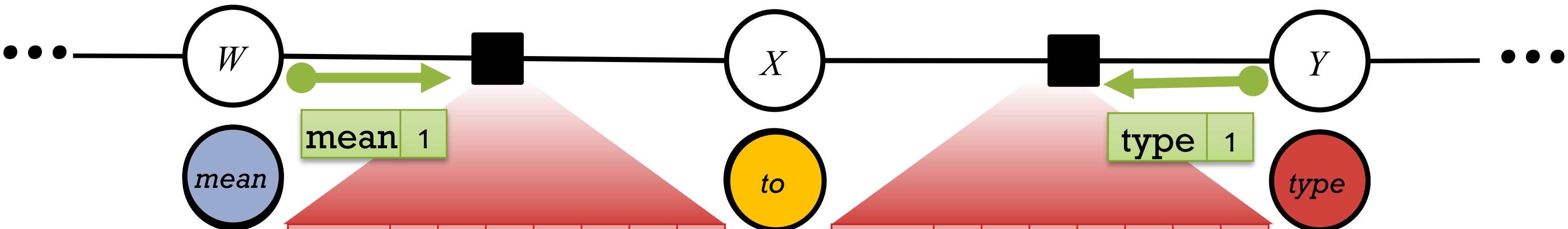
From Gibbs Sampling to Particle BP to BP



Approach 1: Gibbs Sampling

- For each variable, resample the value by conditioning on all the other variables
  - Called the "full conditional" distribution
  - Computationally easy because we really only need to condition on the Markov Blanket
- We can view the computation of the full conditional in terms of message passing
  - **Message puts all its probability mass on the current particle (i.e. current *value*)**

From Gibbs Sampling to Particle BP to BP



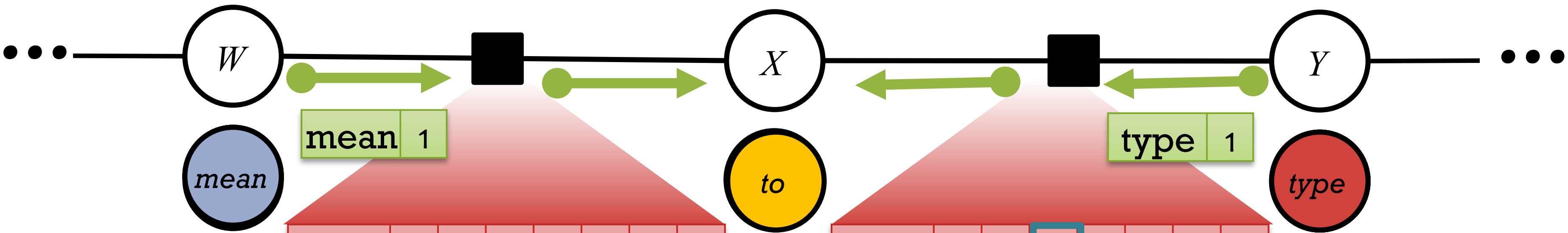
	abacus	…	to	too	two	…	zythum
abacus	0.1		0.2	0.1	0.1		0.1
…							
man	0.1		2	4	0.1		0.1
mean	0.1		7	1	2		0.1
meant	0.2		8	1	3		0.1
…							
zythum	0.1		0.1	0.2	0.2		0.1

	abacus	…	type	tight	taipei	…	zythum
abacus	0.1		0.1	0.2	0.1		0.1
…							
to	0.2		8	3	2		0.1
too	0.1		7	6	1		0.1
two	0.2		0.1	3	1		0.1
…							
zythum	0.1		0.2	0.2	0.1		0.1

Approac… …g

- For each v… …e by co… …variables
  - Called … …n
  - Computationally easy because we really only need to condition on the Markov Blanket
- We can view the computation of the full conditional in terms of message passing
  - Message puts all its probability mass on the current particle (i.e. current *value*)

From Gibbs Sampling to Particle BP to BP



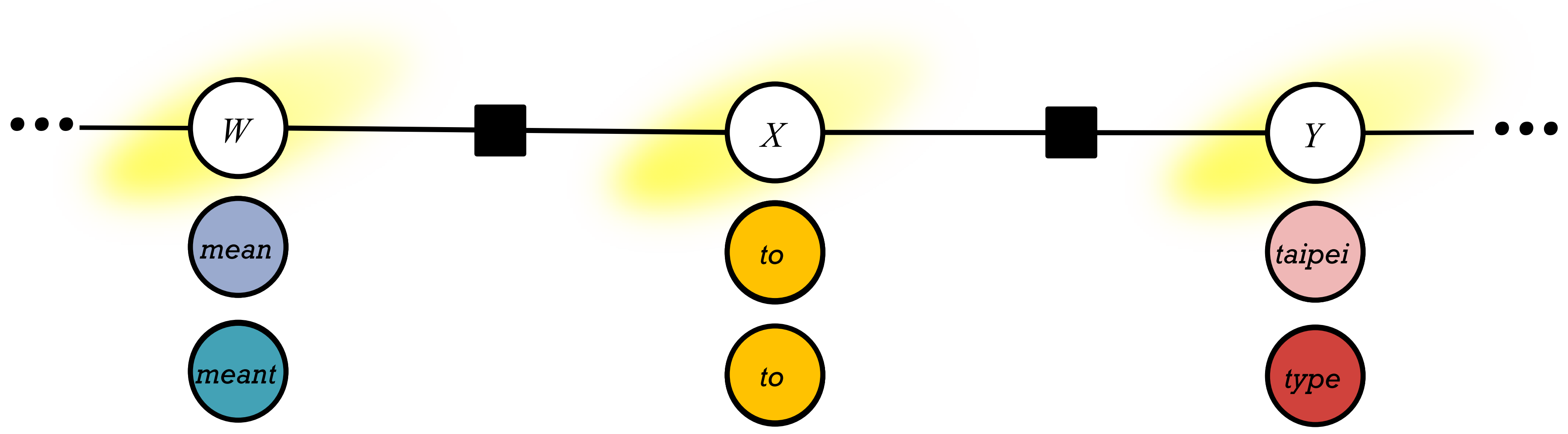
	abacus	…	to	too	two	…	zythum
abacus	0.1		0.2	0.1	0.1		0.1
…							
man	0.1		2	4	0.1		0.1
mean	0.1		7	1	2		0.1
meant	0.2		8	1	3		0.1
…							
zythum	0.1		0.1	0.2	0.2		0.1

	abacus	…	type	tight	taipei	…	zythum
abacus	0.1		0.1	0.2	0.1		0.1
…							
to	0.2		8	3	2		0.1
too	0.1		7	6	1		0.1
two	0.2		0.1	3	1		0.1
…							
zythum	0.1		0.2	0.2	0.1		0.1

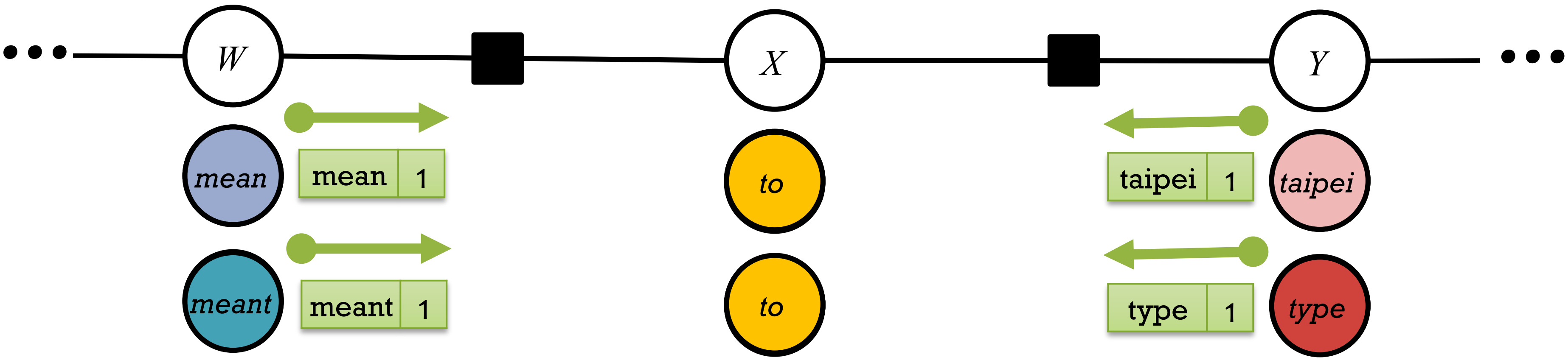
Approac… g

- For each v… by co… variables
  - Called … h
  - Computationally easy because we really only need to condition on the Markov Blanket
- We can view the computation of the full conditional in terms of message passing
  - **Message puts all its probability mass on the current particle (i.e. current *value*)**

From Gibbs Sampling to Particle BP to BP



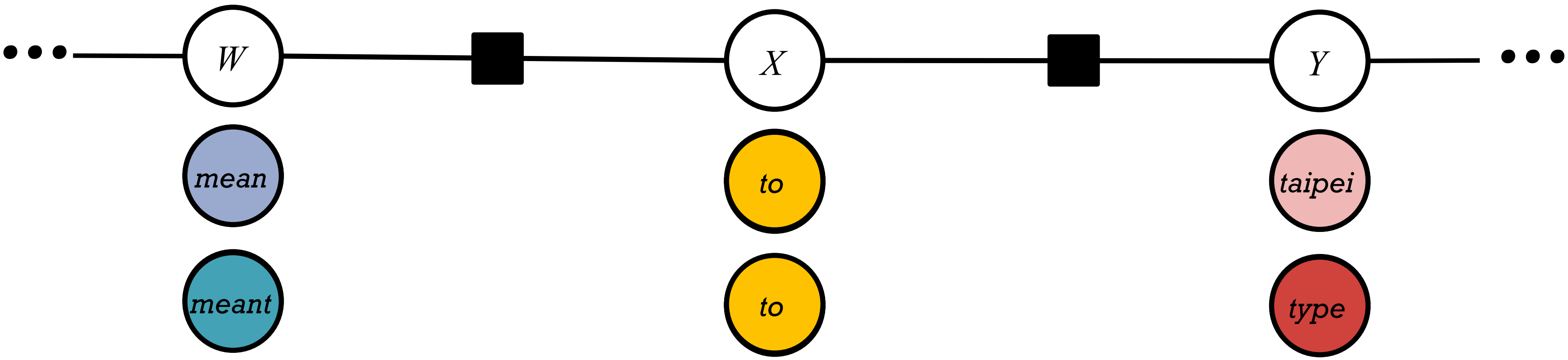
From Gibbs Sampling to Particle BP to BP



Approach 2: Multiple Gibbs Samplers

- Run each Gibbs Sampler independently
- Full conditionals computed independently
  - k separate messages that are each a pointmass distribution

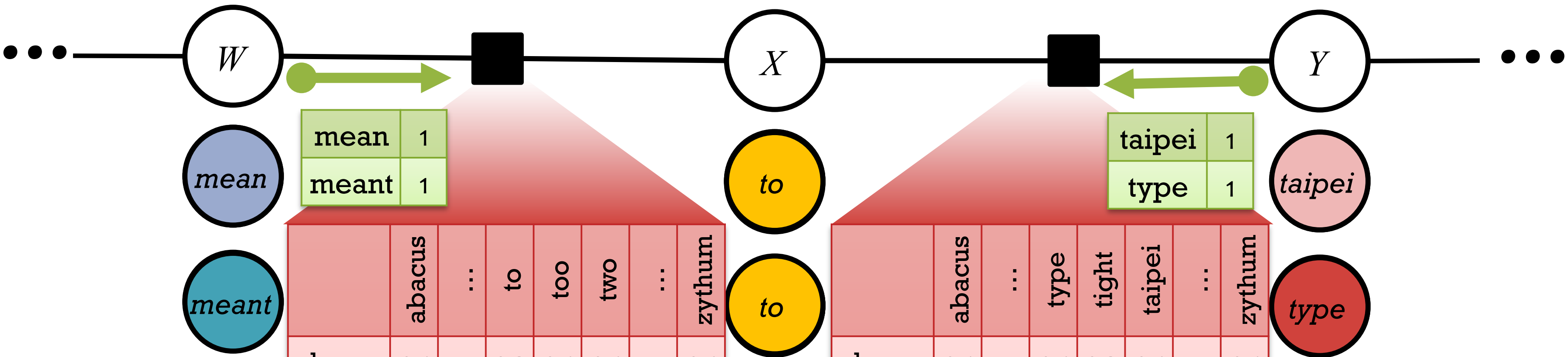
From Gibbs Sampling to Particle BP to BP



Approach 3: Gibbs Sampling w/Averaging

- Keep k samples for each variable
- Resample from the **average of the full conditionals** for each possible pair of variables
  - **Message is a uniform distribution over current particles**

From Gibbs Sampling to Particle BP to BP



mean	1
meant	1

	abacus	…	to	too	two	…	zythum
abacus	0.1		0.2	0.1	0.1		0.1
…							
man	0.1		2	4	0.1		0.1
mean	0.1		7	1	2		0.1
meant	0.2		8	1	3		0.1
…							
zythum	0.1		0.1	0.2	0.2		0.1

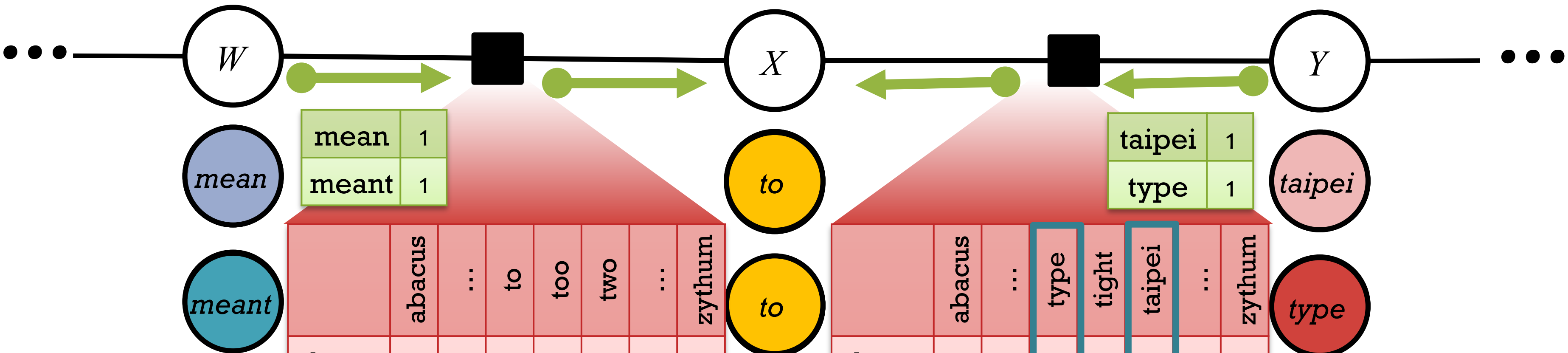
taipei	1
type	1

	abacus	…	type	tight	taipei	…	zythum
abacus	0.1		0.1	0.2	0.1		0.1
…							
to	0.2		8	3	2		0.1
too	0.1		7	6	1		0.1
two	0.2		0.1	3	1		0.1
…							
zythum	0.1		0.2	0.2	0.1		0.1

Approac… g w/…

- Keep k … able
- Resample from the **average of the full conditionals** for each possible pair of variables
  - Message is a uniform distribution over current particles

From Gibbs Sampling to Particle BP to BP



mean	1
meant	1

	abacus	…	to	too	two	…	zythum
abacus	0.1		0.2	0.1	0.1		0.1
…							
man	0.1		2	4	0.1		0.1
mean	0.1		7	1	2		0.1
meant	0.2		8	1	3		0.1
…							
zythum	0.1		0.1	0.2	0.2		0.1

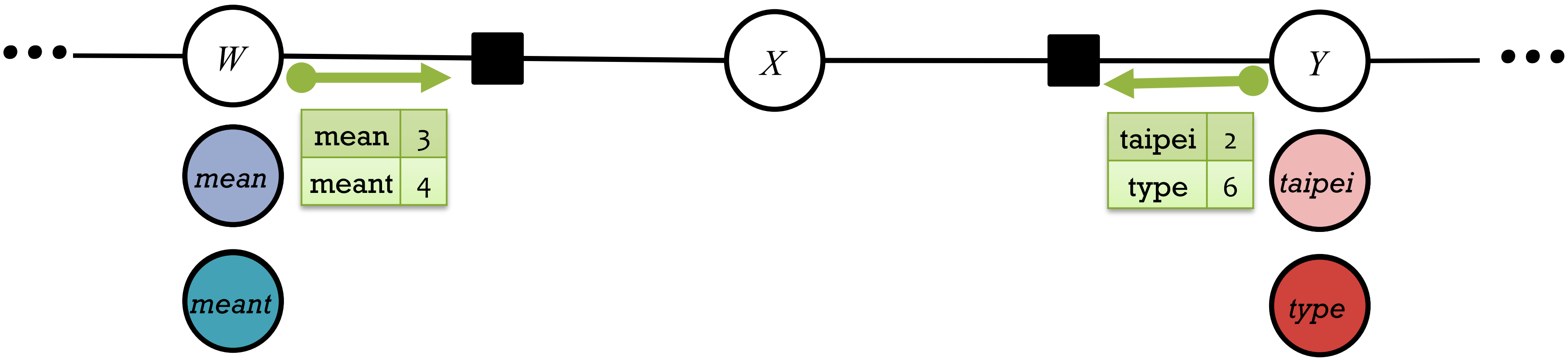
taipei	1
type	1

	abacus	…	type	tight	taipei	…	zythum
abacus	0.1		0.1	0.2	0.1		0.1
…							
to	0.2		8	3	2		0.1
too	0.1		7	6	1		0.1
two	0.2		0.1	3	1		0.1
…							
zythum	0.1		0.2	0.2	0.1		0.1

Approac… g w/…

- Keep k … able
- Resample from the **average of the full conditionals** for each possible pair of variables
  - Message is a uniform distribution over current particles

From Gibbs Sampling to Particle BP to BP

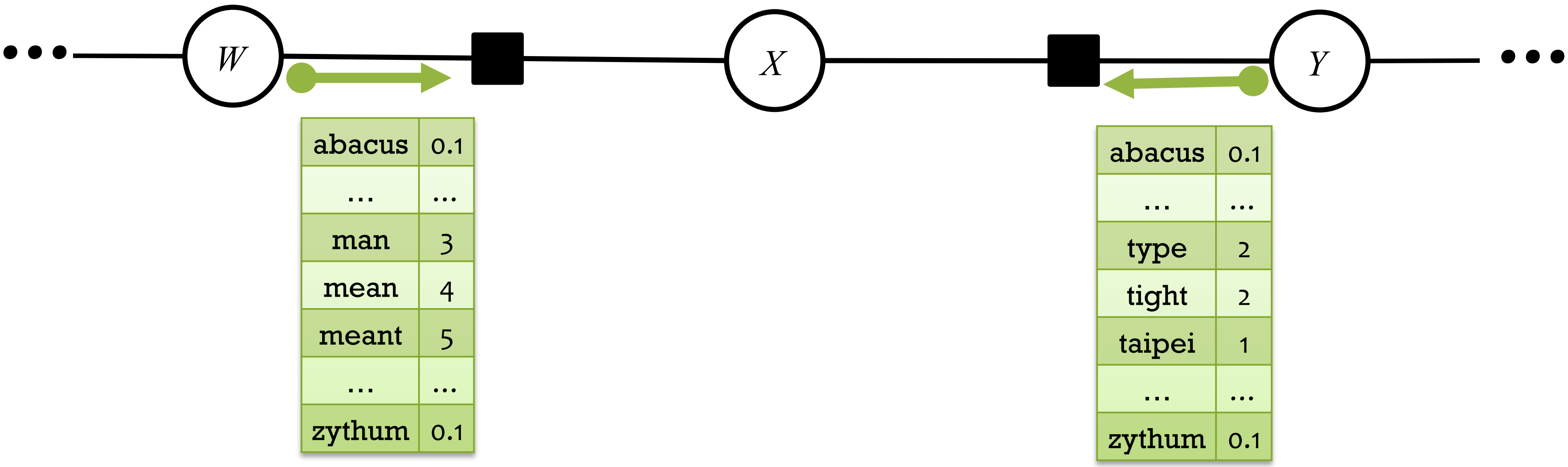


Approach 4: Particle BP

- Similar in spirit to Gibbs Sampling w/Averaging
- **Messages are a weighted distribution over *k* particles**

(Ihler & McAllester, 2009)

From Gibbs Sampling to Particle BP to BP



Approach 5: BP

- In Particle BP, as the number of particles goes to $+\infty$, the estimated messages approach the true BP messages
- **Belief propagation represents messages as the full distribution**
  - This assumes we can **store** the whole distribution compactly

(Ihler & McAllester, 2009)

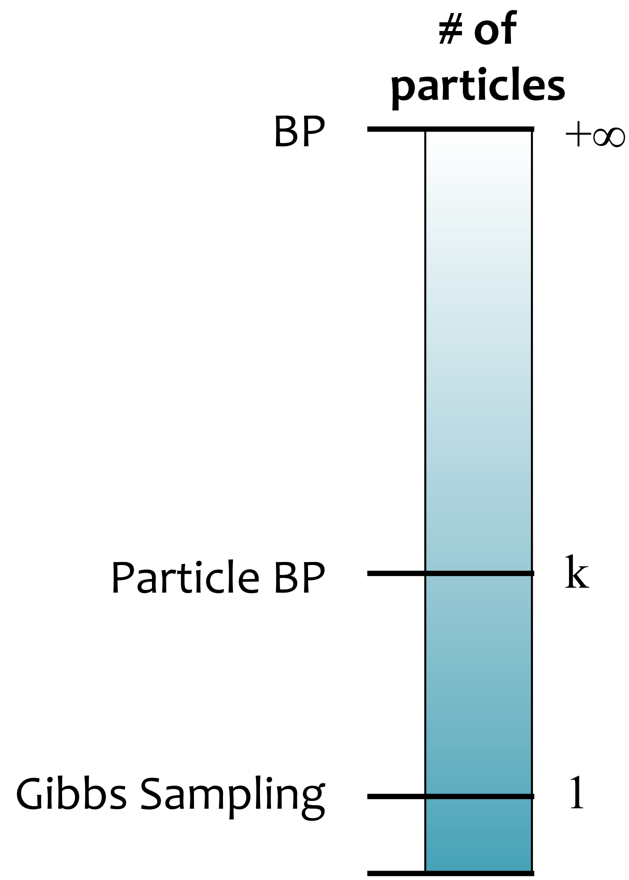
From Gibbs Sampling to Particle BP to BP

Message Representation:

A. Belief Propagation: **full distribution**

B. Gibbs sampling: **single particle**

C. Particle BP: **multiple particles**



From Gibbs Sampling to Particle BP to BP

Tension between approaches…

Sampling values or combinations of values:

- quickly get a good estimate of the frequent cases
- may take a long time to estimate probabilities of infrequent cases
- may take a long time to draw a sample (mixing time)
- exact if you run forever

Enumerating each value and computing its probability exactly:

- have to spend time on all values
- but only spend O(1) time on each value (don't sample frequent values over and over while waiting for infrequent ones)
- runtime is more predictable
- lets you tradeoff exactness for greater speed (brute force exactly enumerates exponentially many assignments, BP approximates this by enumerating local configurations)

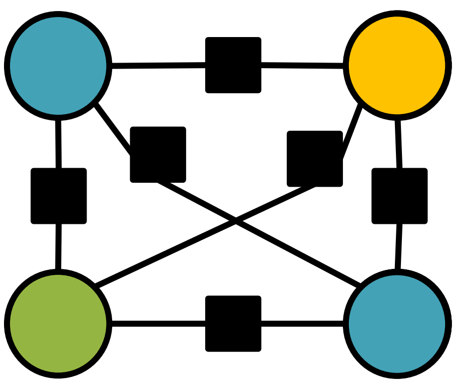
Background: Convergence

When BP is run on a **tree-shaped factor graph**, the beliefs **converge** to the marginals of the distribution after two passes.

Q&A

Q: How **long** does loopy BP take to **converge**?

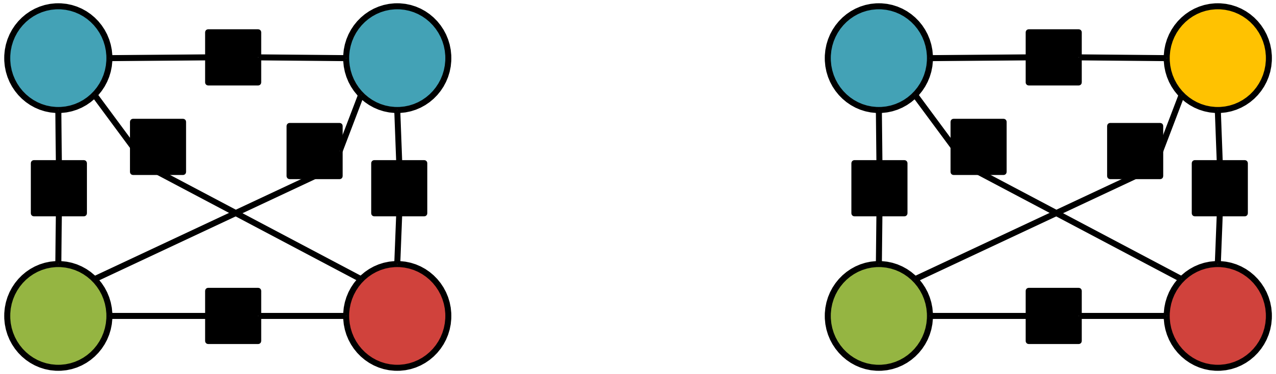
A: It might never converge. Could oscillate.



Q&A

Q: When loopy BP converges, does it always get the same answer?

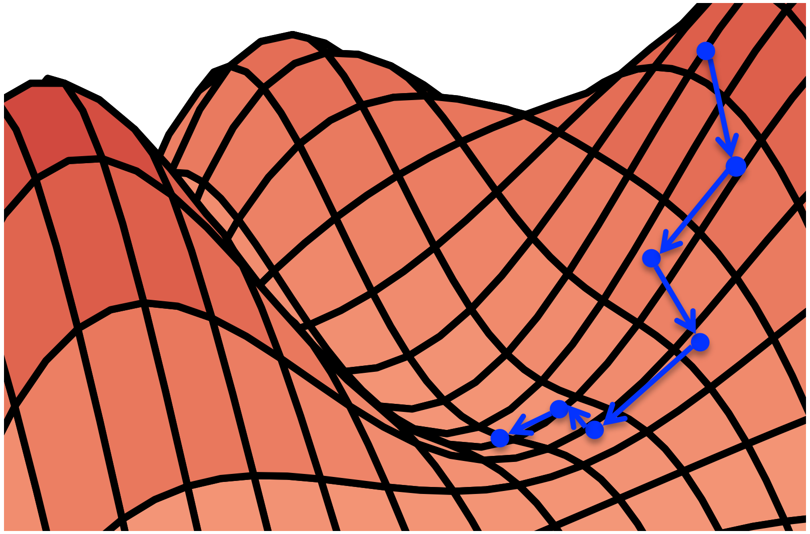
A: No. Sensitive to initialization and update order.



Q&A

Q: Are there convergent variants of loopy BP?

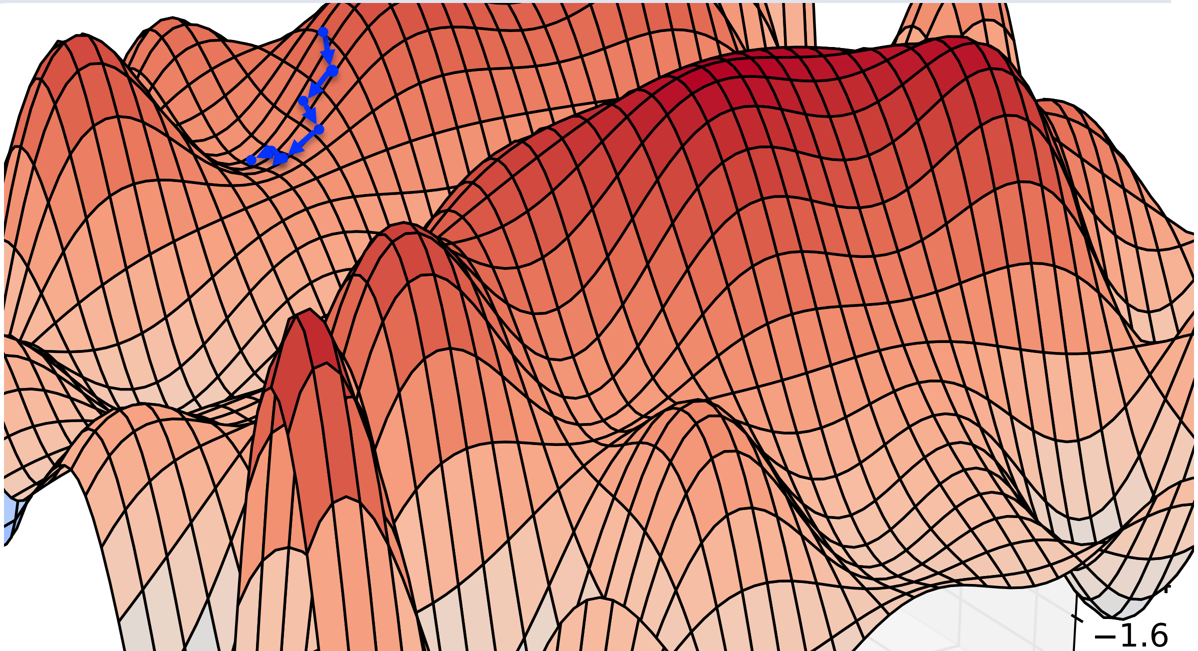
A: Yes. It's actually trying to minimize a certain **differentiable** function of the beliefs, so you could just **minimize** that function **directly**.



Q&A

Q: But does that function have a unique minimum?

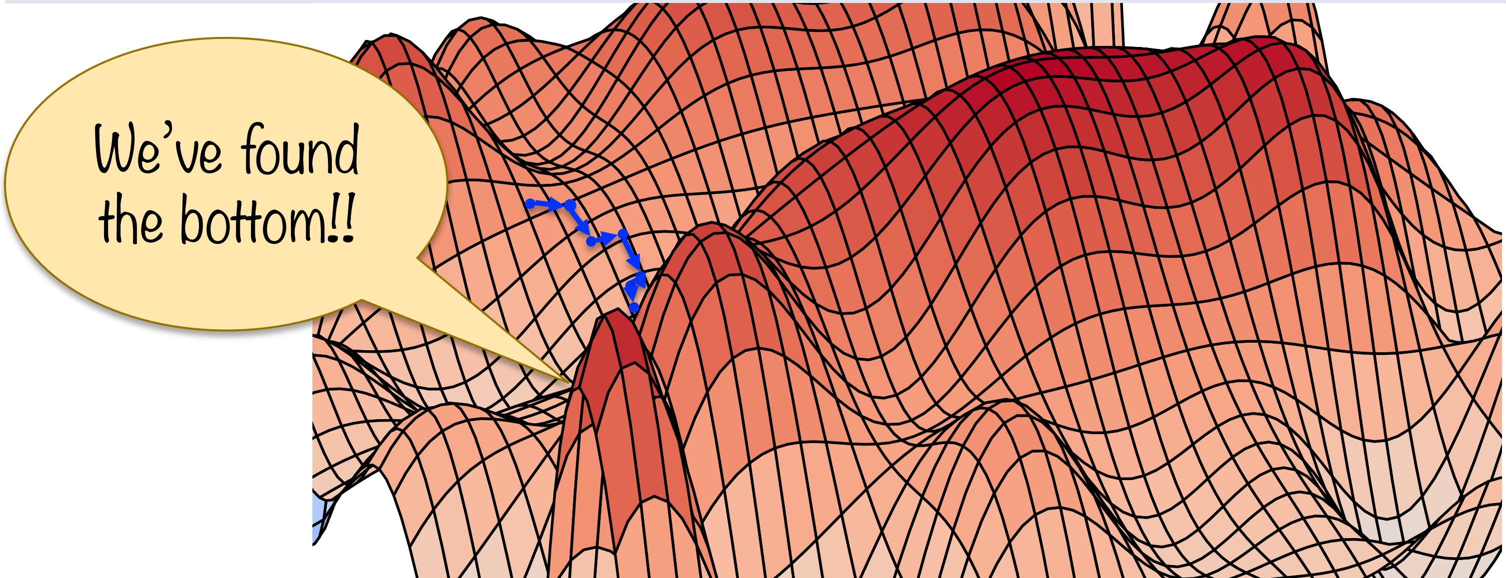
A: No, and you'll only be able to find a local minimum in practice. So you're still dependent on initialization.



Q&A

Q: If you could find the global minimum, would its beliefs give the marginals of the **true distribution**?

A: No.

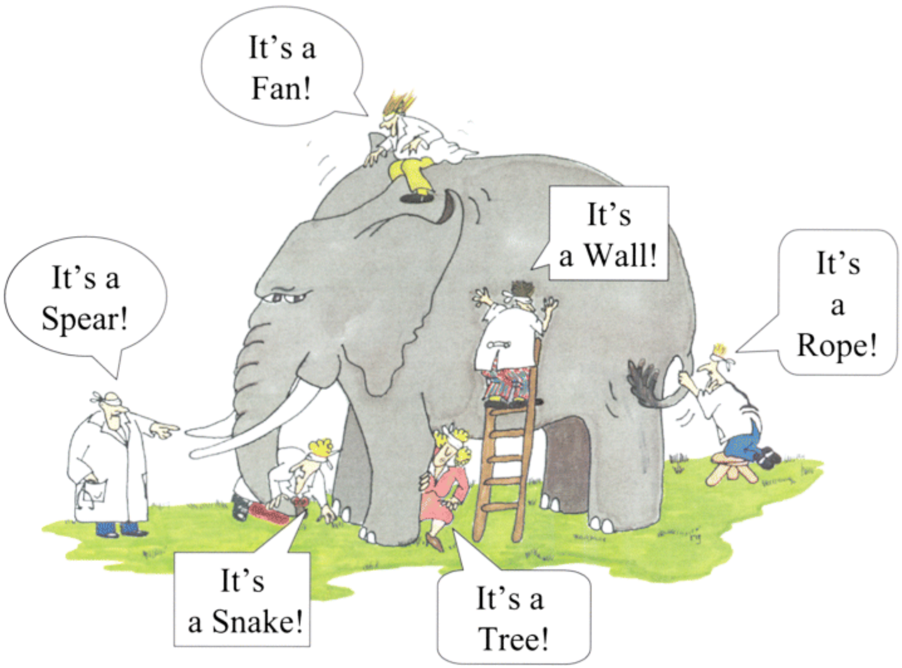


Q&A

Q: Is it finding the marginals of some **other distribution** (as mean field would)?

A: No, just a **collection of beliefs.**

Might not be globally consistent in the sense of all being views of the same elephant.



*Cartoon by G. Renee Guzlas

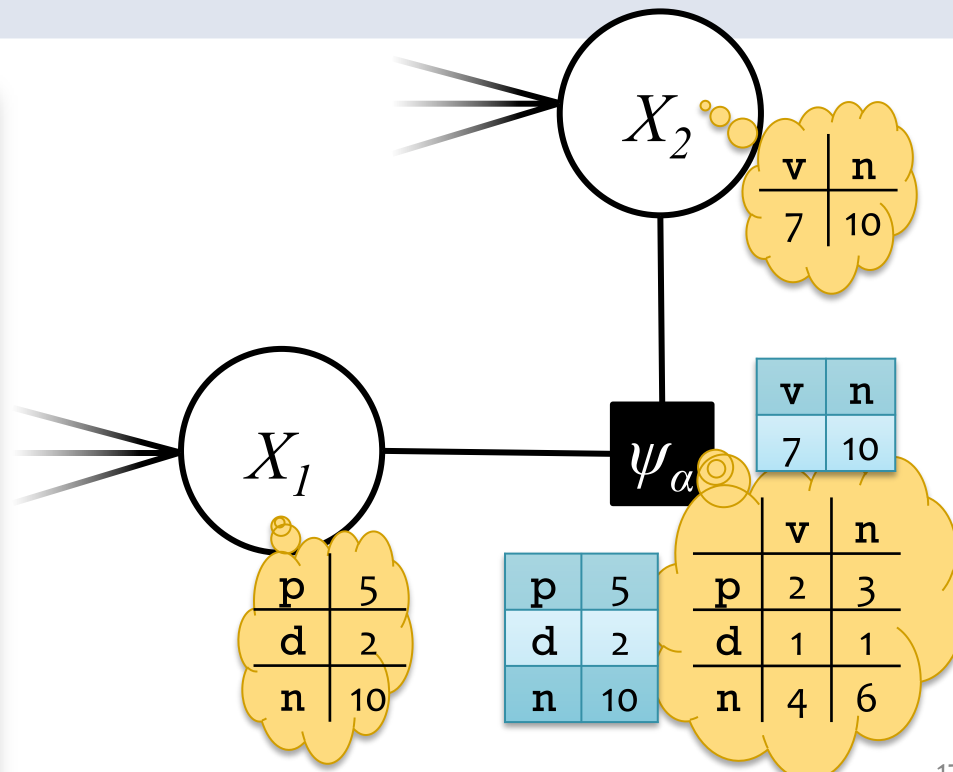
Q&A

Q: Does the global minimum give beliefs that are at least **locally consistent**?

A: Yes.

A variable belief and a factor belief are <u>locally consistent</u> if the marginal of the factor's belief equals the variable's belief.

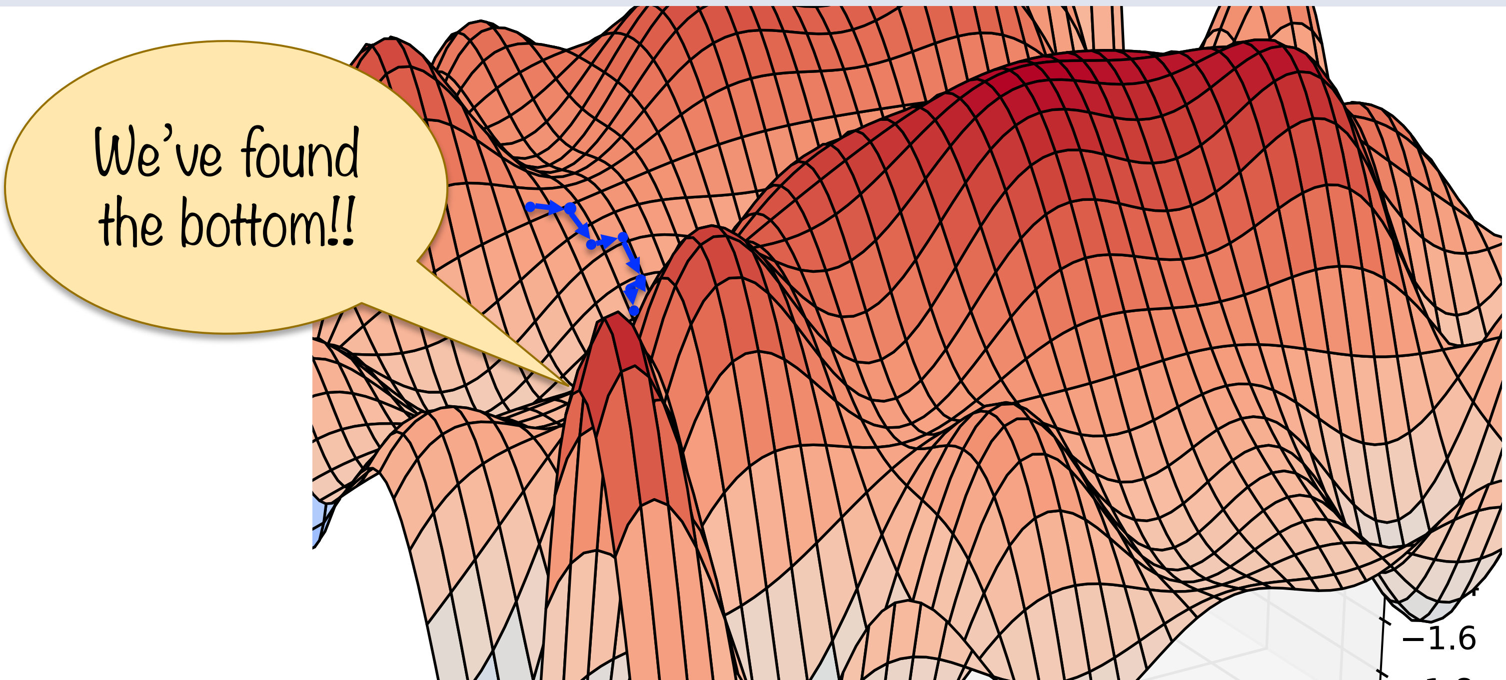
$$b_i(x_i) = \sum_{\boldsymbol{x}_{\boldsymbol{\alpha}} \setminus x_i} b_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}}), \quad \forall i, \alpha \in \mathcal{N}(i)$$



Q&A

Q: In what sense are the beliefs at the **global minimum** any good?

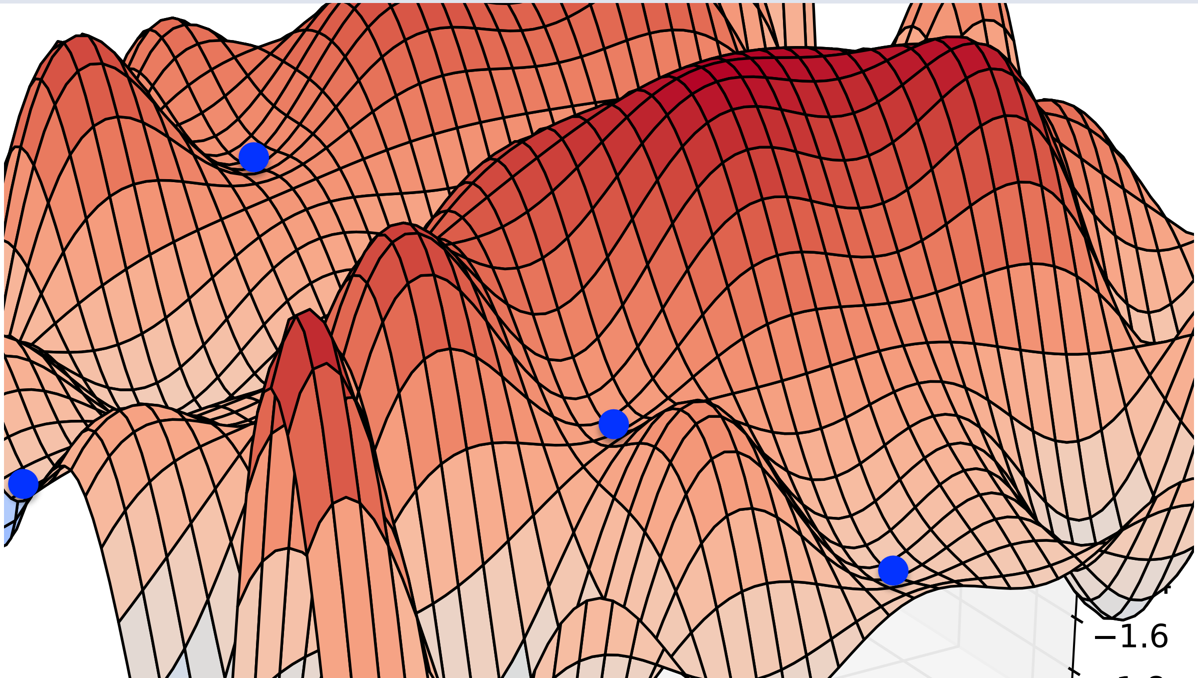
A: They are the global minimum of the **Bethe Free Energy**.



Q&A

Q: When loopy BP **converges**, in what sense are the **beliefs** any good?

A: They are a **local minimum** of the **Bethe Free Energy**.



Q&A

Q: Why would you want to minimize the Bethe Free Energy?

A:
1) It's easy to minimize* because it's a sum of functions on the individual beliefs.
2) On an *acyclic* factor graph, it measures KL divergence between beliefs and true marginals, and so is minimized when beliefs = marginals. (For a *loopy* graph, we close our eyes and hope it still works.)

[*] Though we can't just minimize each function separately – we need message passing to keep the beliefs locally consistent.

Section 3: Appendix

BP as an Optimization Algorithm

BP as an Optimization Algorithm

This Appendix provides a more in-depth study of BP as an optimization algorithm.

Our focus is on the Bethe Free Energy and its relation to KL divergence, Gibbs Free Energy, and the Helmholtz Free Energy.

We also include a discussion of the convergence properties of max-product BP.

KL and Free Energies

Kullback–Leibler (KL) divergence

$$\mathrm{KL}(b||p) = \sum_{\boldsymbol{x}} b(\boldsymbol{x}) \log \left[\frac{b(\boldsymbol{x})}{p(\boldsymbol{x})} \right]$$

$$= \sum_{\boldsymbol{x}} b(\boldsymbol{x}) \log \left[\frac{b(\boldsymbol{x})}{\prod_{\alpha} \psi_{\alpha}(\boldsymbol{x}_{\boldsymbol{\alpha}})} \right] + \log Z$$

Gibbs Free Energy

$$F(b) = \mathrm{KL}(b||p) - \log Z = \sum_{\boldsymbol{x}} b(\boldsymbol{x}) \log \left[\frac{b(\boldsymbol{x})}{\prod_{\alpha} \psi_{\alpha}(\boldsymbol{x}_{\boldsymbol{\alpha}})} \right]$$

Helmholtz Free Energy

$$F_H = -\log Z = \min_{b} F(b)$$

Minimizing KL Divergence

- If we find the distribution $\boldsymbol{b}$ that minimizes the KL divergence, then $\boldsymbol{b} = \boldsymbol{p}$

$$p(\boldsymbol{x}) = \operatorname*{argmin}_b \mathrm{KL}(b||p)$$

$$= \operatorname*{argmin}_b \sum_{\boldsymbol{x}} b(\boldsymbol{x}) \log \left[\frac{b(\boldsymbol{x})}{\prod_\alpha \psi_\alpha(\boldsymbol{x_\alpha})} \right]$$

- Also, true of the minimum of the Gibbs Free Energy
- But what if $\boldsymbol{b}$ is not (necessarily) a probability distribution?

BP on a 2 Variable Chain

True distribution:

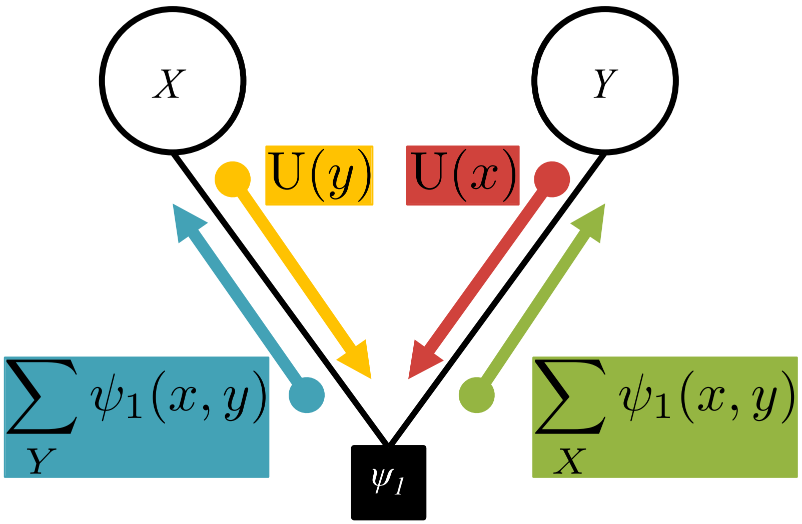
$$p(x,y) = \frac{\psi_1(x,y)}{Z}$$

Beliefs at the end of BP:

$$b(x,y) = \frac{\psi_1(x,y)}{Z}$$

$$b(x) \propto \sum_Y \psi_1(x,y)$$

$$b(y) \propto \sum_X \psi_1(x,y)$$

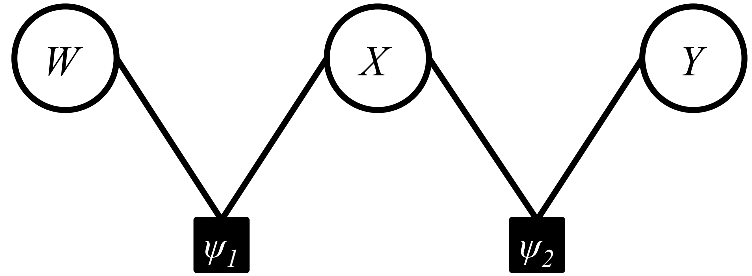


We successfully minimized the KL divergence!

$$p(\boldsymbol{x}) = \operatorname*{argmin}_b \mathrm{KL}(b||p)$$

*where U(x) is the uniform distribution

BP on a 3 Variable Chain



True distribution:

$$p(w,x,y) = \frac{\psi_1(w,x)\psi_2(x,y)}{Z}$$

The true distribution can be expressed in terms of its **marginals**:

$$p(w,x,y) = p(w|x)p(x,y)$$
$$= \frac{p(w,x)p(x,y)}{p(x)}$$

Define the **joint belief** to have the same form:

$$b(w,x,y) := \frac{b(w,x)b(x,y)}{b(x)}$$

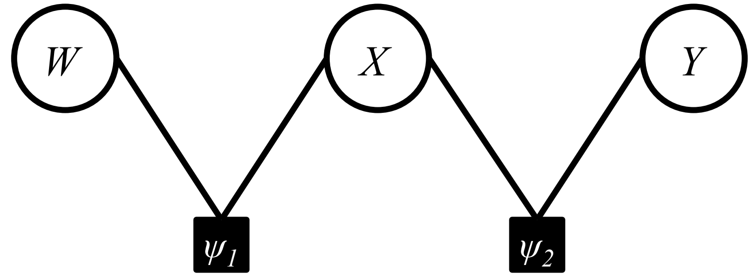
$$\begin{aligned}\mathrm{KL}(b||p) &= \sum_{w,x,y} b(w,x,y)\log\left[\frac{b(w,x,y)}{p(w,x,y)}\right] \\ &= \sum_{w,x} b(w,x)\log\left[\frac{b(w,x)}{\psi_1(w,x)}\right] \\ &\quad + \sum_{x,y} b(x,y)\log\left[\frac{b(x,y)}{\psi_2(x,y)}\right] \\ &\quad - \sum_{x} b(x)\log b(x) + \log Z\end{aligned}$$

KL decomposes over the marginals

BP on a 3 Variable Chain

True distribution:

$$p(w,x,y) = \frac{\psi_1(w,x)\psi_2(x,y)}{Z}$$



The true distribution can be expressed in terms of its **marginals**:

$$p(w,x,y) = p(w|x)p(x,y)$$
$$= \frac{p(w,x)p(x,y)}{p(x)}$$

Define the **joint belief** to have the same form:

$$b(w,x,y) := \frac{b(w,x)b(x,y)}{b(x)}$$

$$F(b) = \sum_{w,x} b(w,x) \log \left[\frac{b(w,x)}{\psi_1(w,x)} \right] + \sum_{x,y} b(x,y) \log \left[\frac{b(x,y)}{\psi_2(x,y)} \right] - \sum_{x} b(x) \log b(x)$$

Gibbs Free Energy decomposes over the marginals

BP on an Acyclic Graph

True distribution:

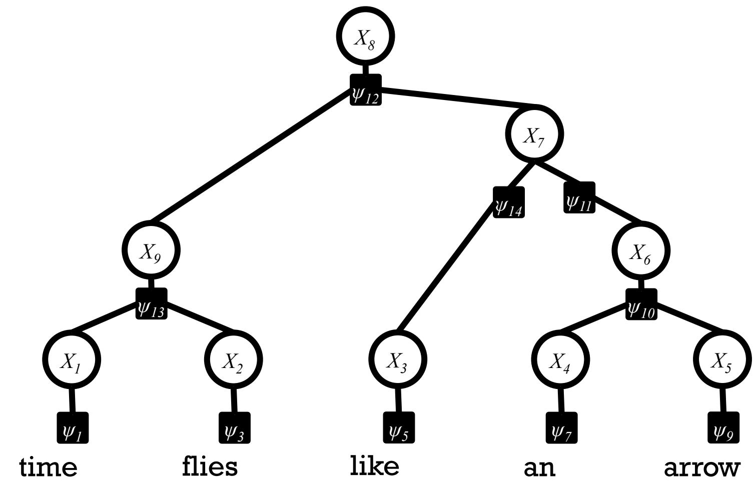
$$p(\boldsymbol{x}) = \frac{1}{Z} \prod_{\alpha} \psi_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}})$$

The true distribution can be expressed in terms of its **marginals**:

$$p(\boldsymbol{x}) = \frac{\prod_\alpha p(\boldsymbol{x}_{\boldsymbol{\alpha}})}{\prod_i p(x_i)^{N_i - 1}}$$

Define the **joint belief** to have the same form:

$$b(\boldsymbol{x}) := \frac{\prod_\alpha b_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}})}{\prod_i b_i(x_i)^{N_i - 1}}$$



$$\begin{aligned}
\mathrm{KL}(b||p) &= \sum_{\boldsymbol{x}} b(\boldsymbol{x}) \log \left[\frac{b(\boldsymbol{x})}{p(\boldsymbol{x})} \right] \\
&= \underbrace{\sum_{\alpha} \sum_{\boldsymbol{x}_{\boldsymbol{\alpha}}} b_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}}) \log \left[\frac{b_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}})}{\psi_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}})} \right] - \sum_i (N_i - 1) \sum_{x_i} b_i(x_i) \log b_i(x_i) + \log Z}
\end{aligned}$$

KL decomposes over the marginals

BP on an Acyclic Graph

True distribution:

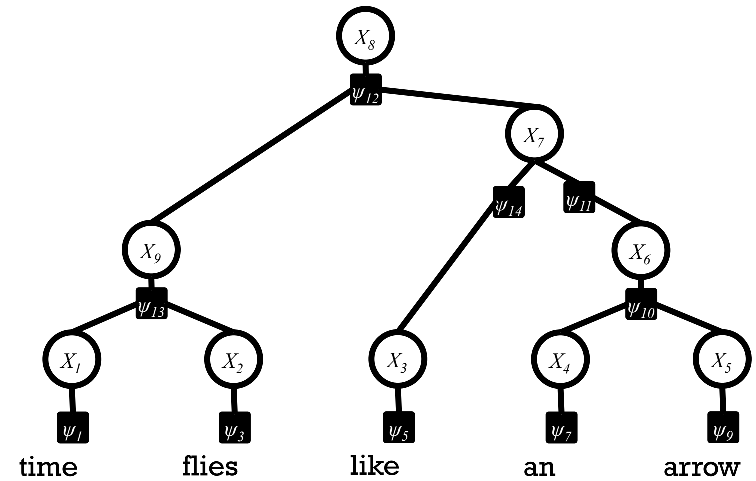
$$p(\boldsymbol{x}) = \frac{1}{Z} \prod_{\alpha} \psi_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}})$$

The true distribution can be expressed in terms of its **marginals**:

$$p(\boldsymbol{x}) = \frac{\prod_\alpha p(\boldsymbol{x}_{\boldsymbol{\alpha}})}{\prod_i p(x_i)^{N_i - 1}}$$

Define the **joint belief** to have the same form:

$$b(\boldsymbol{x}) := \frac{\prod_\alpha b_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}})}{\prod_i b_i(x_i)^{N_i - 1}}$$



$$F(b) = \sum_\alpha \sum_{\boldsymbol{x}_{\boldsymbol{\alpha}}} b_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}}) \log \left[\frac{b_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}})}{\psi_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}})} \right] \underbrace{- \sum_i (N_i - 1) \sum_{x_i} b_i(x_i) \log b_i(x_i)}$$

Gibbs Free Energy decomposes over the marginals

BP on a Loopy Graph

True distribution:

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

Construct the joint belief as before:

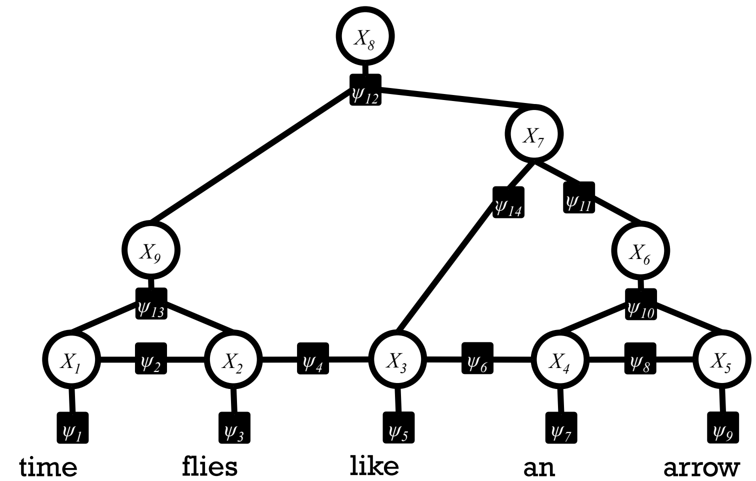
$$b(\boldsymbol{x}) := \frac{\prod_\alpha b_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}})}{\prod_i b_i(x_i)^{N_i - 1}}$$

This might **not** be a distribution!

So add constraints…

1. The beliefs are distributions: are non-negative and sum-to-one.
2. The beliefs are locally consistent:

$$b_i(x_i) = \sum_{\boldsymbol{x}_{\boldsymbol{\alpha}} \setminus x_i} b_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}}), \quad \forall i, \alpha \in \mathcal{N}(i)$$



KL is no longer well defined, because the joint belief is not a proper distribution.

$$\cancel{\mathrm{KL}(b||p) = \sum_{w,x,y} b(w,x,y) \log \left[\frac{b(w,x,y)}{p(w,x,y)} \right]}$$

BP on a Loopy Graph

True distribution:

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

Construct the **joint belief** as before:

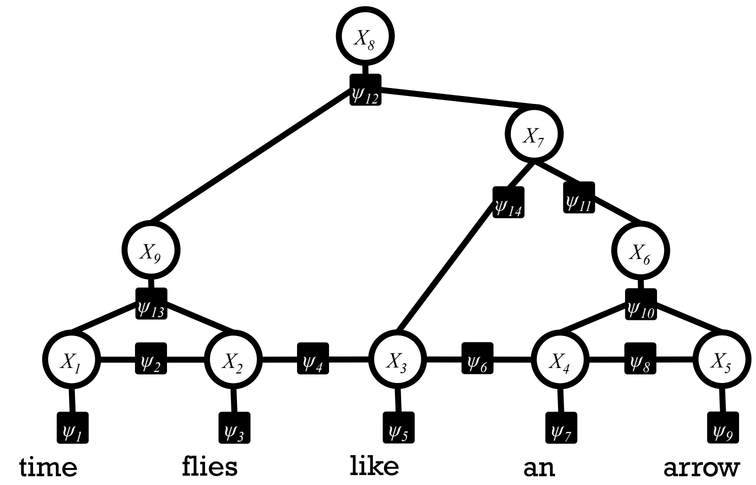
$$b(\boldsymbol{x}) := \frac{\prod_\alpha b_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}})}{\prod_i b_i(x_i)^{N_i - 1}}$$

This might **not** be a distribution!

So add constraints…

1. The beliefs are distributions: are non-negative and sum-to-one.
2. The beliefs are locally consistent:

$$b_i(x_i) = \sum_{\boldsymbol{x}_{\boldsymbol{\alpha}} \setminus x_i} b_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}}), \quad \forall i, \alpha \in \mathcal{N}(i)$$



But we can still optimize the same objective as before, subject to our belief constraints:

$$F_{\text{Bethe}}(b) = \sum_\alpha \sum_{\boldsymbol{x}_{\boldsymbol{\alpha}}} b_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}}) \log \left[\frac{b_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}})}{\psi_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}})} \right] - \sum_i (N_i - 1) \sum_{x_i} b_i(x_i) \log b_i(x_i)$$

This is called the **Bethe Free Energy** and decomposes over the marginals

BP as an Optimization Algorithm

- The **Bethe Free Energy**, a function of the beliefs:

$$F_{\text{Bethe}}(b) = \sum_{\alpha} \sum_{\boldsymbol{x}_{\boldsymbol{\alpha}}} b_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}}) \log \left[\frac{b_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}})}{\psi_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}})} \right] - \sum_i (N_i - 1) \sum_{x_i} b_i(x_i) \log b_i(x_i)$$

- BP minimizes a **constrained** version of the Bethe Free Energy
  - BP is just one local optimization algorithm: fast but not guaranteed to converge
  - If BP converges, the beliefs are called **fixed points**
  - The **stationary points** of a function have a gradient of zero

The **fixed points** of BP are local **stationary points** of the Bethe Free Energy (Yedidia, Freeman, & Weiss, 2000)

BP as an Optimization Algorithm

- The **Bethe Free Energy,** a function of the beliefs:

$$F_{\text{Bethe}}(b) = \sum_{\alpha} \sum_{\boldsymbol{x}_{\boldsymbol{\alpha}}} b_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}}) \log \left[\frac{b_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}})}{\psi_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}})} \right]$$
$$- \sum_i (N_i - 1) \sum_{x_i} b_i(x_i) \log b_i(x_i)$$

- BP minimizes a **constrained** version of the Bethe Free Energy
  - BP is just one local optimization algorithm: fast but not guaranteed to converge
  - If BP converges, the beliefs are called **fixed points**
  - The **stationary points** of a function have a gradient of zero

The **stable fixed points** of BP are local **minima** of the Bethe Free Energy (Heskes, 2003)

BP as an Optimization Algorithm

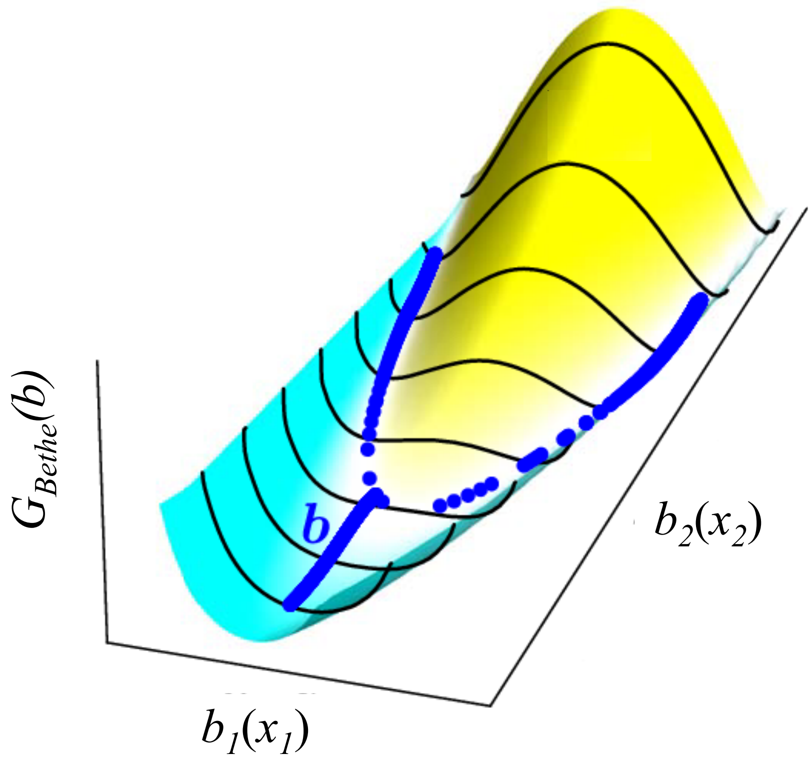
For graphs with no cycles:

- The minimizing beliefs = the true marginals
- BP finds the global minimum of the Bethe Free Energy
- This **global minimum** is $-\log Z$ (the "Helmholtz Free Energy")

For graphs with cycles:

- The minimizing beliefs only **approximate** the true marginals
- Attempting to minimize may get stuck at **local minimum** or other critical point
- Even the global minimum only approximates $-\log Z$

Convergence of Sum-product BP



The figure shows a two-dimensional slice of the Bethe Free Energy for a binary graphical model with pairwise interactions

- The fixed point beliefs:
  - Do not necessarily correspond to marginals of any joint distribution over all the variables (Mackay, Yedidia, Freeman, & Weiss, 2001; Yedidia, Freeman, & Weiss, 2005)
- *Unbelievable* probabilities
  - Conversely, the true marginals for many joint distributions cannot be reached by BP (Pitkow, Ahmadian, & Miller, 2011)

Figure adapted from (Pitkow, Ahmadian, & Miller, 2011)

Convergence of Max-product BP

If the max-marginals $b_i(x_i)$ are a fixed point of BP, and $\boldsymbol{x}^*$ is the corresponding assignment (assumed unique), then $p(\boldsymbol{x}^*) > p(x)$ for every $x \neq \boldsymbol{x}^*$ in a rather large neighborhood around $\boldsymbol{x}^*$ (Weiss & Freeman, 2001).

The **neighbors of $\boldsymbol{x}^*$** are constructed as follows: For any set of vars S of **disconnected trees and single loops**, set the variables in S to arbitrary values, and the rest to x^*.

Informally: If you take the fixed-point solution $\boldsymbol{x}^*$ and arbitrarily change the values of the dark nodes in the figure, the overall probability of the configuration will decrease.

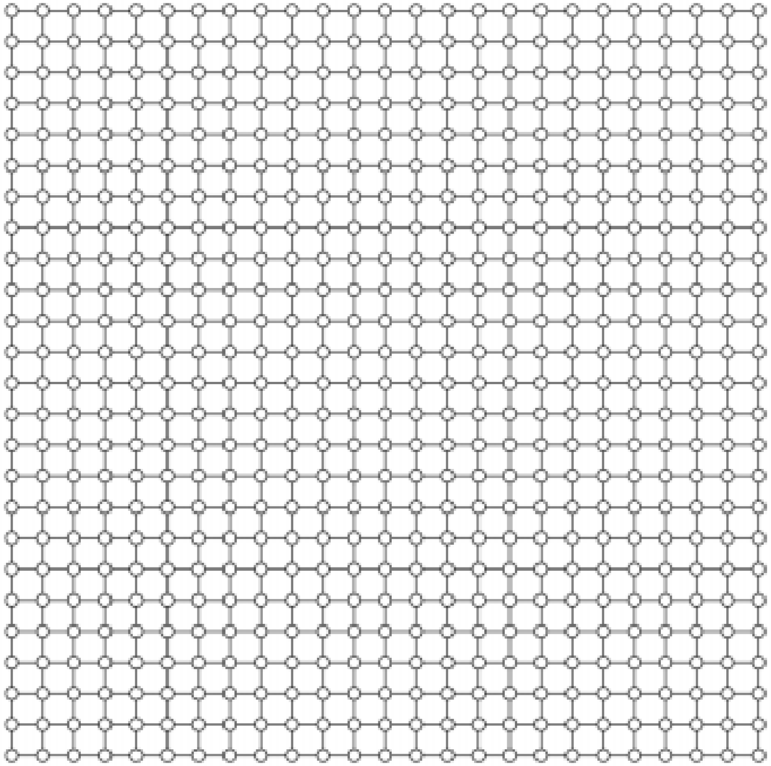


Figure from (Weiss & Freeman, 2001)

Convergence of Max-product BP

If the max-marginals $b_i(x_i)$ are a fixed point of BP, and $\boldsymbol{x}^*$ is the corresponding assignment (assumed unique), then $p(\boldsymbol{x}^*) > p(x)$ for every $x \neq \boldsymbol{x}^*$ in a rather large neighborhood around $\boldsymbol{x}^*$ (Weiss & Freeman, 2001).

The **neighbors of $\boldsymbol{x}^*$** are constructed as follows: For any set of vars S of **disconnected trees and single loops**, set the variables in S to arbitrary values, and the rest to x^*.

Informally: If you take the fixed-point solution $\boldsymbol{x}^*$ and arbitrarily change the values of the dark nodes in the figure, the overall probability of the configuration will decrease.

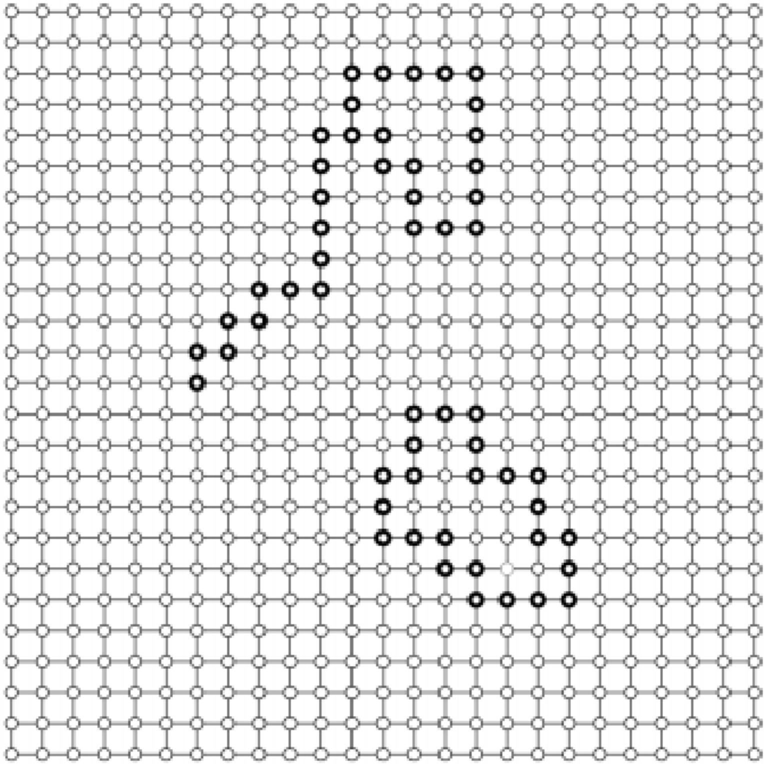


Figure from (Weiss & Freeman, 2001)

Convergence of Max-product BP

If the max-marginals $b_i(x_i)$ are a fixed point of BP, and $\boldsymbol{x}^*$ is the corresponding assignment (assumed unique), then $p(\boldsymbol{x}^*) > p(x)$ for every $x \neq \boldsymbol{x}^*$ in a rather large neighborhood around $\boldsymbol{x}^*$ (Weiss & Freeman, 2001).

The **neighbors of** $\boldsymbol{x}^*$ are constructed as follows: For any set of vars S of **disconnected trees and single loops**, set the variables in S to arbitrary values, and the rest to x^*.

Informally: If you take the fixed-point solution $\boldsymbol{x}^*$ and arbitrarily change the values of the dark nodes in the figure, the overall probability of the configuration will decrease.

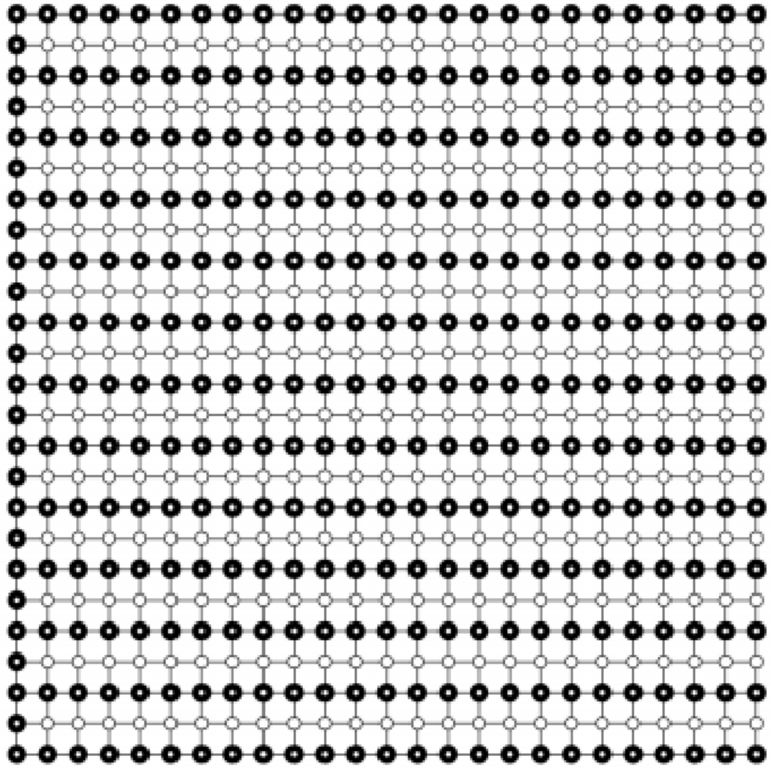


Figure from (Weiss & Freeman, 2001)

Section 4: Incorporating Structure into Factors and Variables

Outline

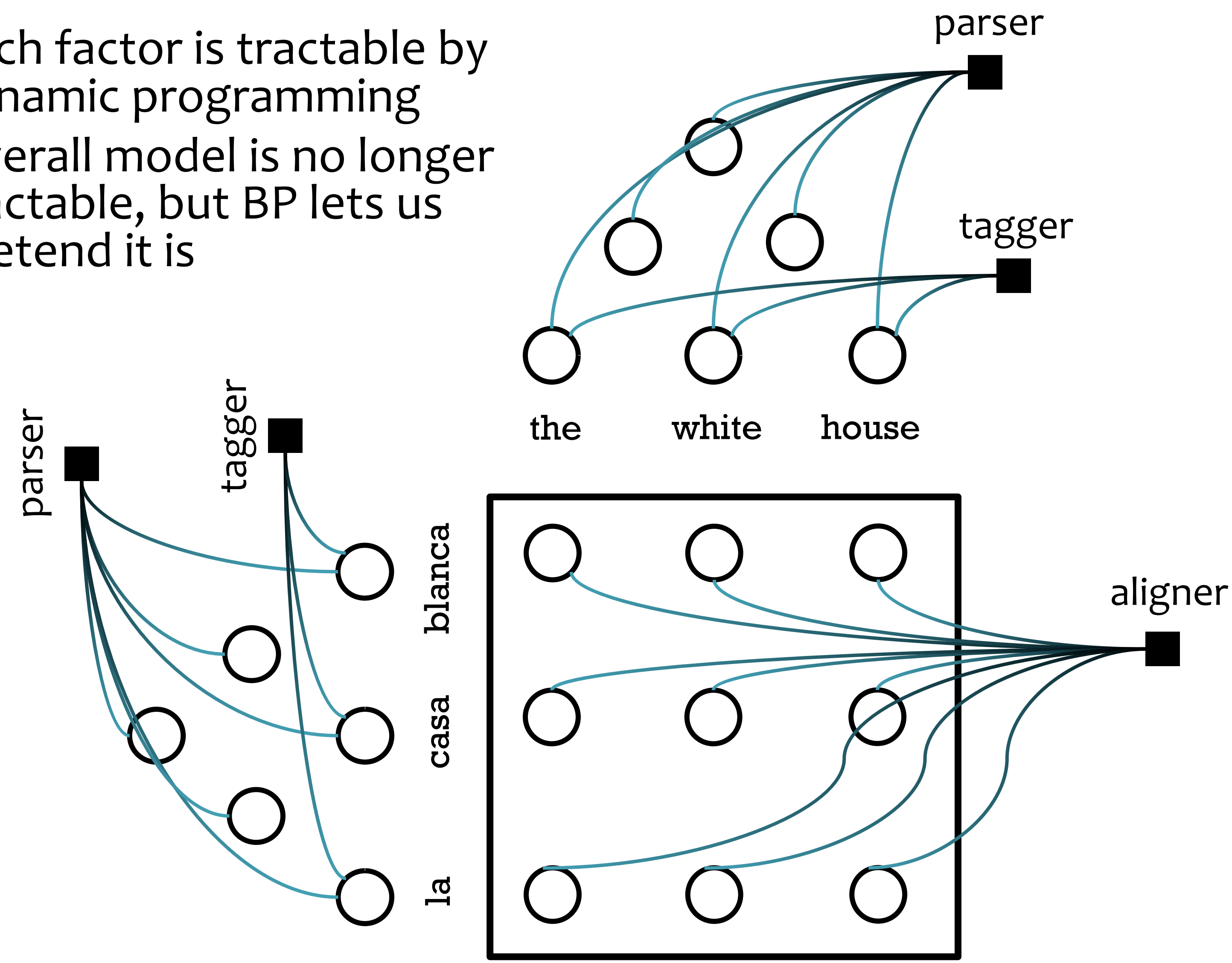
- Do you want to push past the simple NLP models (logistic regression, PCFG, etc.) that we've all been using for 20 years?
- Then this tutorial is extremely practical for you!
  1. **Models:** Factor graphs can express interactions among linguistic structures.
  2. **Algorithm:** BP estimates the global effect of these interactions on each variable, using local computations.
  3. **Intuitions:** What's going on here? Can we trust BP's estimates?
  4. **Fancier Models:** Hide a whole grammar and dynamic programming algorithm within a single factor. BP coordinates multiple factors.
  5. **Tweaked Algorithm:** Finish in fewer steps and make the steps faster.
  6. **Learning:** Tune the parameters. Approximately improve the true predictions -- or truly improve the approximate predictions.
  7. **Software:** Build the model you want!

Outline

- Do you want to push past the simple NLP models (logistic regression, PCFG, etc.) that we've all been using for 20 years?
- Then this tutorial is extremely practical for you!
  1. **Models:** Factor graphs can express interactions among linguistic structures.
  2. **Algorithm:** BP estimates the global effect of these interactions on each variable, using local computations.
  3. **Intuitions:** What's going on here? Can we trust BP's estimates?
  4. **Fancier Models:** Hide a whole grammar and dynamic programming algorithm within a single factor. BP coordinates multiple factors.
  5. **Tweaked Algorithm:** Finish in fewer steps and make the steps faster.
  6. **Learning:** Tune the parameters. Approximately improve the true predictions -- or truly improve the approximate predictions.
  7. **Software:** Build the model you want!

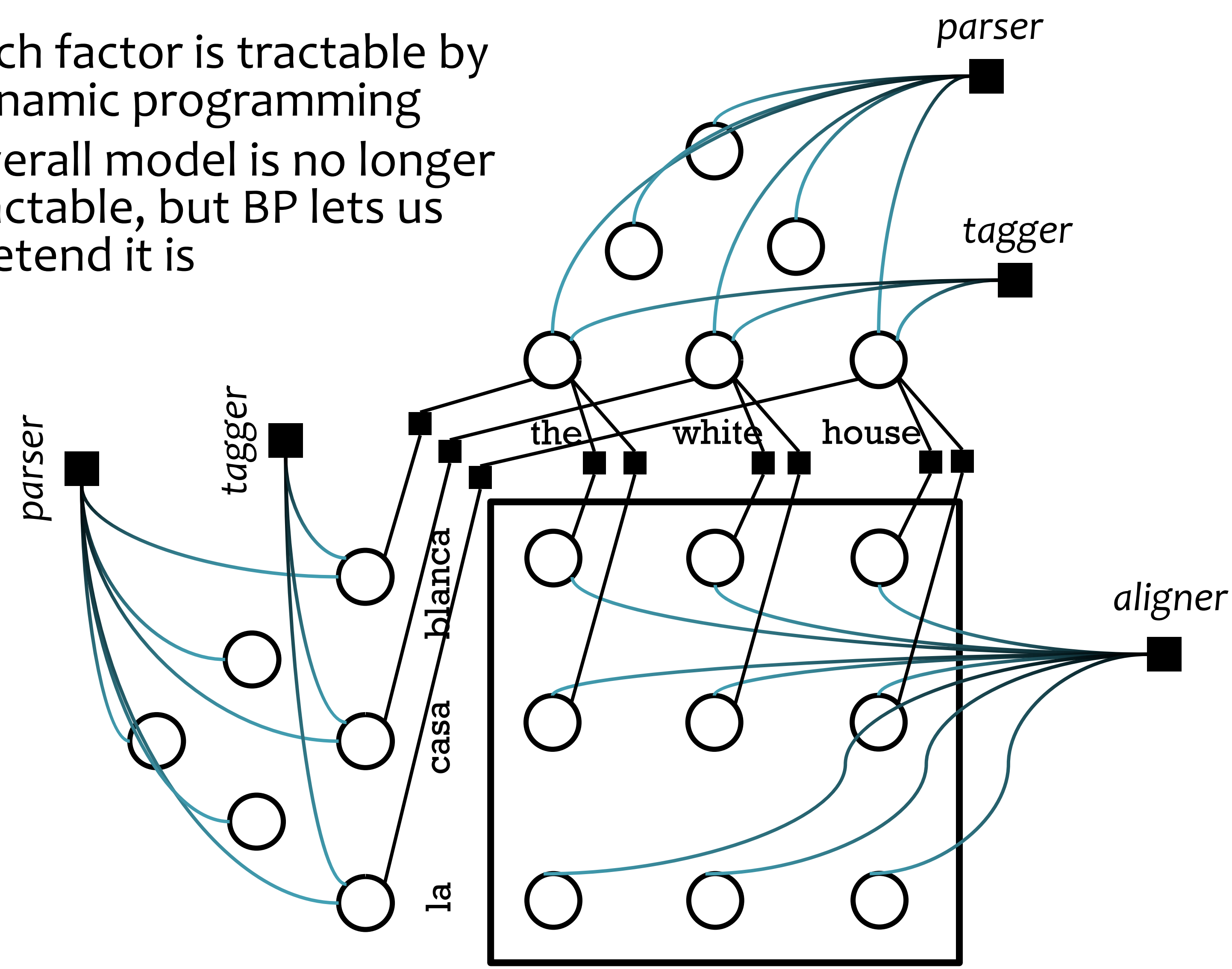
BP for Coordination of Algorithms

- Each factor is tractable by dynamic programming
- Overall model is no longer tractable, but BP lets us pretend it is



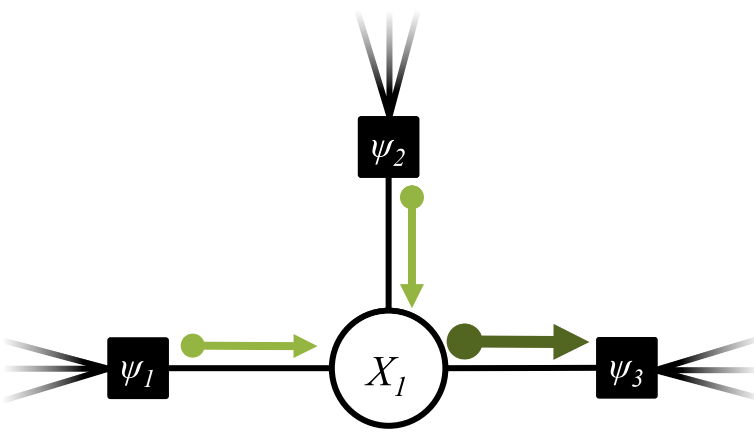
BP for Coordination of Algorithms

- Each factor is tractable by dynamic programming
- Overall model is no longer tractable, but BP lets us pretend it is



Sending Messages: Computational Complexity

From Variables



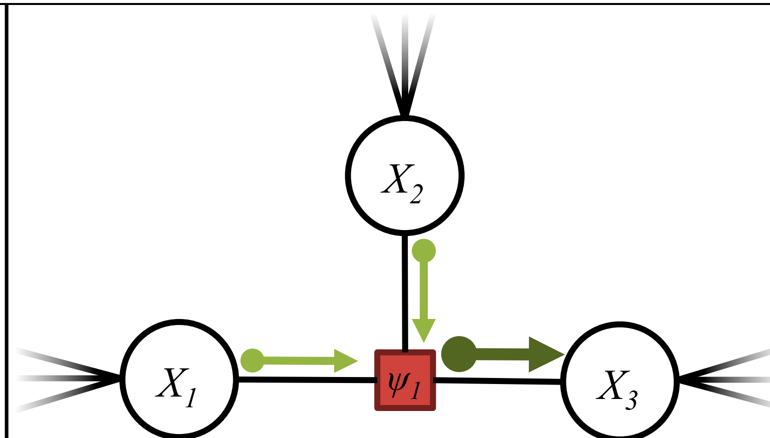
$$\mu_{i \to \alpha}(x_i) = \prod_{\alpha \in \mathcal{N}(i) \setminus \alpha} \mu_{\alpha \to i}(x_i)$$

$O(d^*k)$

d = # of neighboring factors

k = # possible values for X_i

To Variables



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

$O(d^*k^d)$

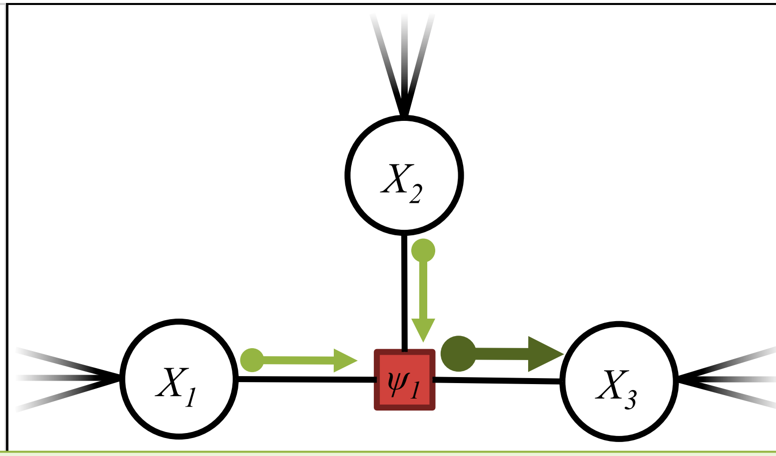
d = # of neighboring variables

k = maximum # possible values for a neighboring variable

Sending Messages: Computational Complexity

From Variables

To Variables



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

$O(d*k)$

d = # of neighboring factors

k = # possible values for X_i

$O(d*k^d)$

d = # of neighboring variables

k = maximum # possible values for a neighboring variable

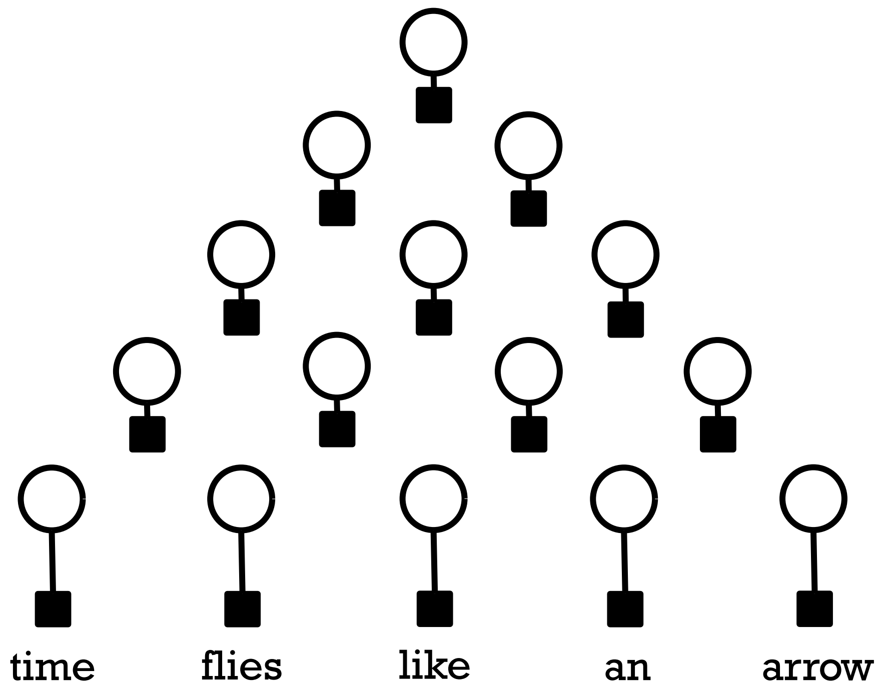
INCORPORATING STRUCTURE INTO FACTORS

Unlabeled Constituency Parsing

Given: a sentence.

Predict: unlabeled parse.

We could predict whether each span is present T or not F.



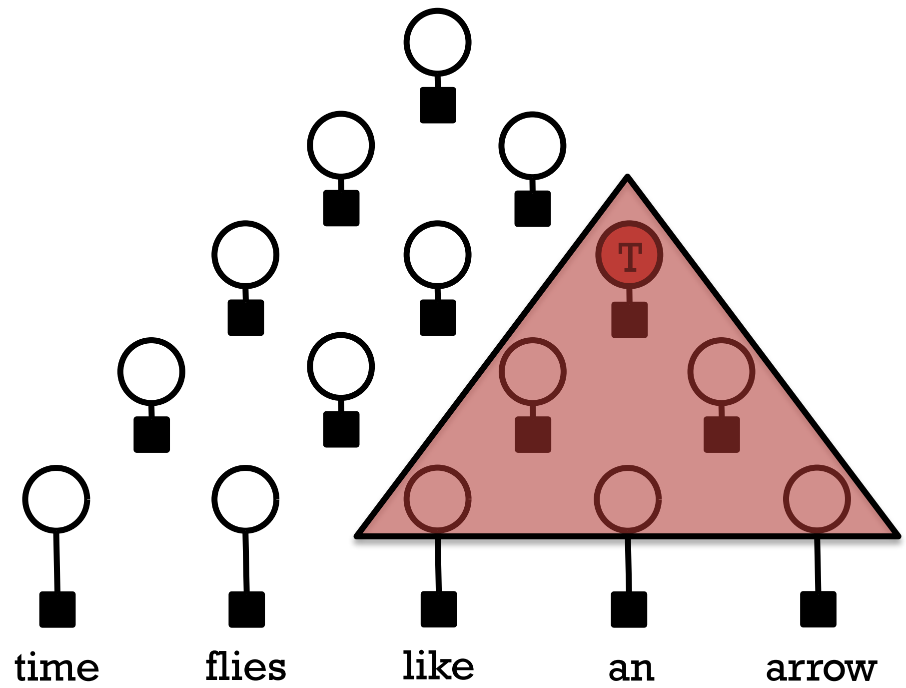
(Naradowsky, Vieira, & Smith, 2012)

Unlabeled Constituency Parsing

Given: a sentence.

Predict: unlabeled parse.

We could predict whether each span is present T or not F.



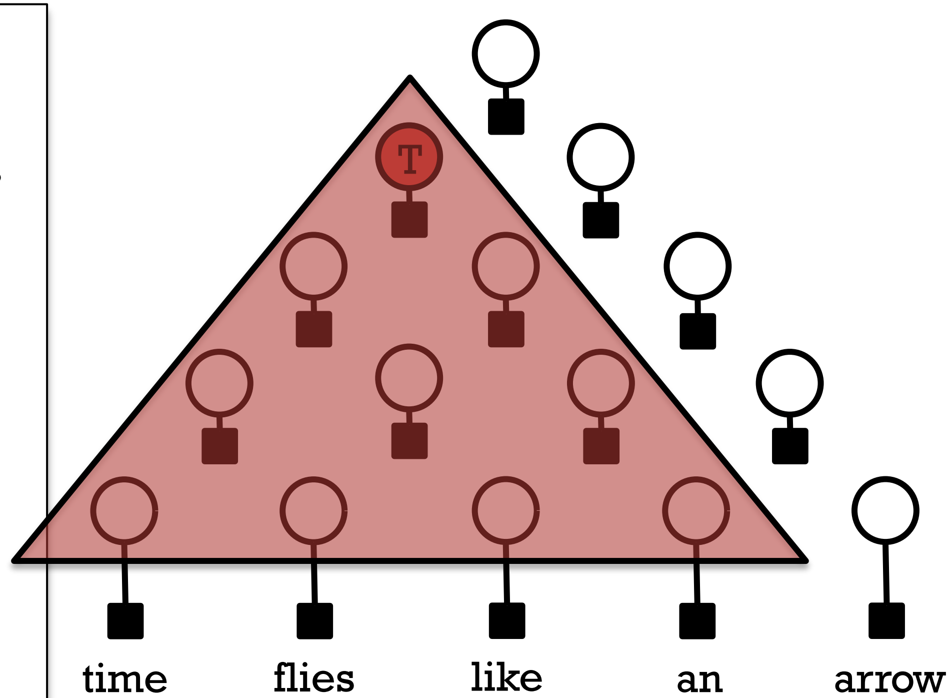
(Naradowsky, Vieira, & Smith, 2012)

Unlabeled Constituency Parsing

Given: a sentence.

Predict: unlabeled parse.

We could predict whether each span is present T or not F.



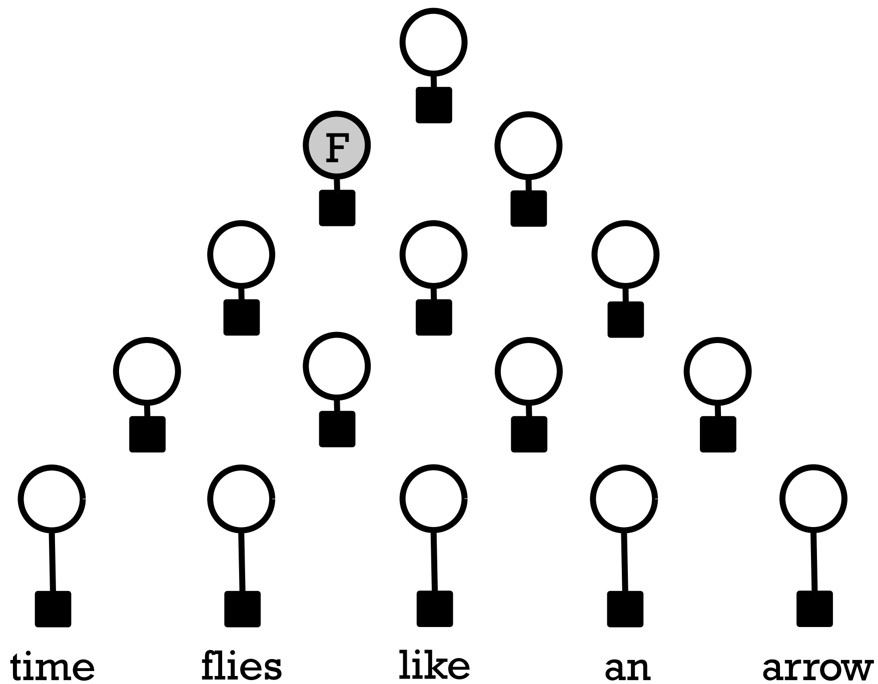
(Naradowsky, Vieira, & Smith, 2012)

Unlabeled Constituency Parsing

Given: a sentence.

Predict: unlabeled parse.

We could predict whether each span is present **T** or not **F**.



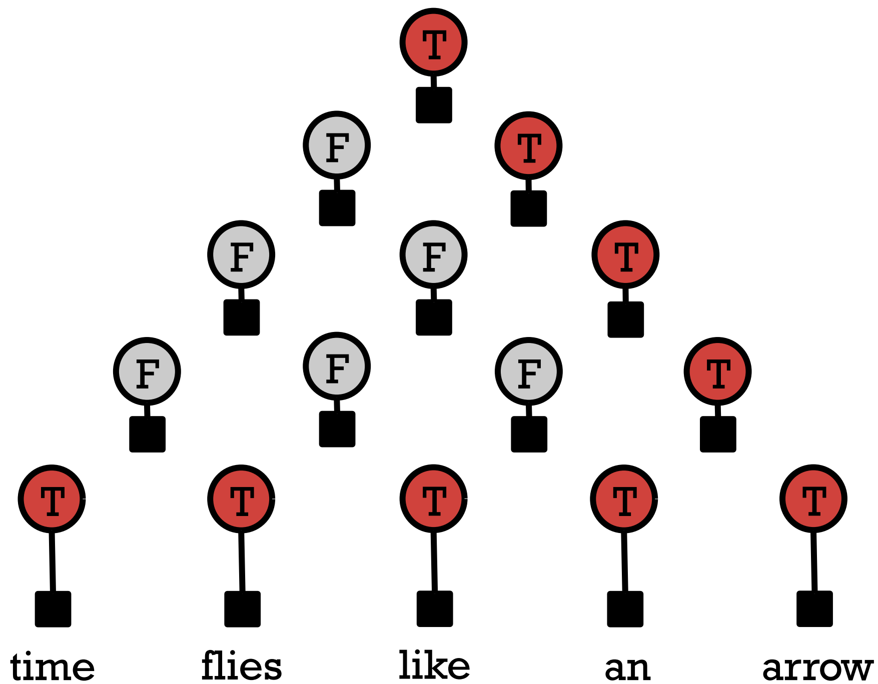
(Naradowsky, Vieira, & Smith, 2012)

Unlabeled Constituency Parsing

Given: a sentence.

Predict: unlabeled parse.

We could predict whether each span is present T or not F.



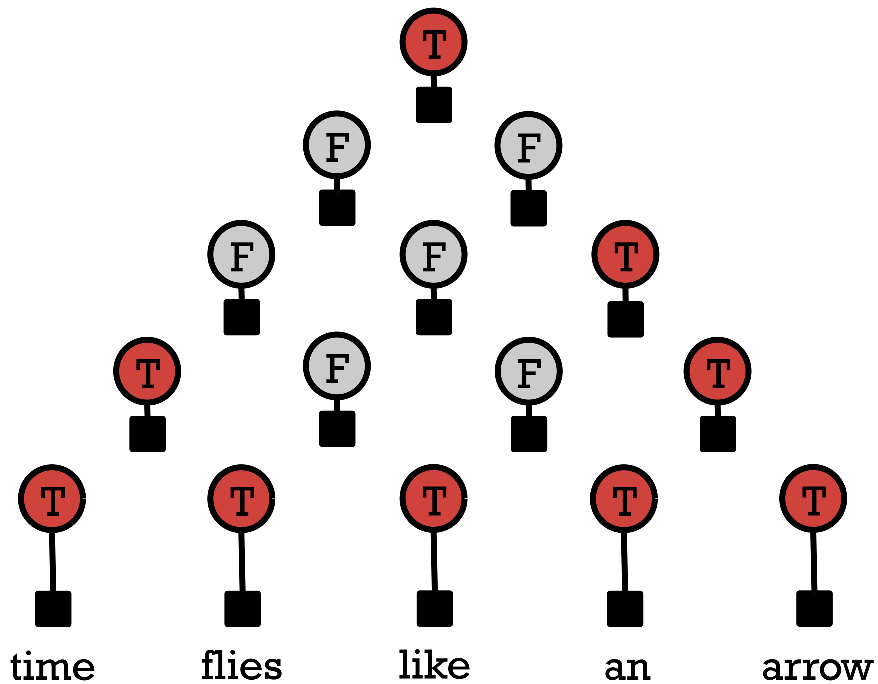
(Naradowsky, Vieira, & Smith, 2012)

Unlabeled Constituency Parsing

Given: a sentence.

Predict: unlabeled parse.

We could predict whether each span is present T or not F.



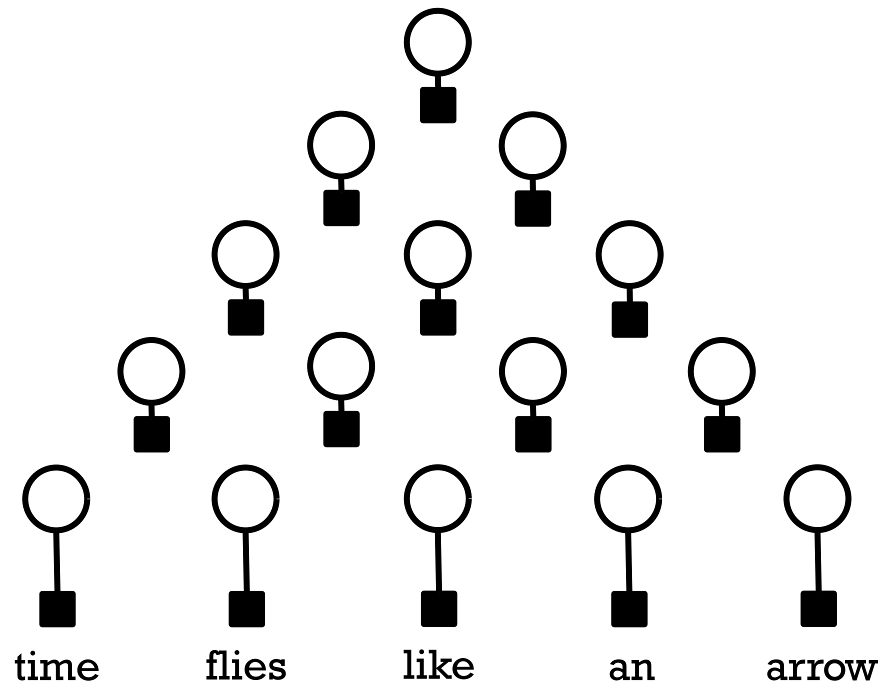
(Naradowsky, Vieira, & Smith, 2012)

Unlabeled Constituency Parsing

Given: a sentence.

Predict: unlabeled parse.

We could predict whether each span is present **T** or not **F**.



Sending a messsage **to a variable** from its unary factors takes only $O(d*k^d)$ time where $k=2$ and $d=1$.

(Naradowsky, Vieira, & Smith, 2012)

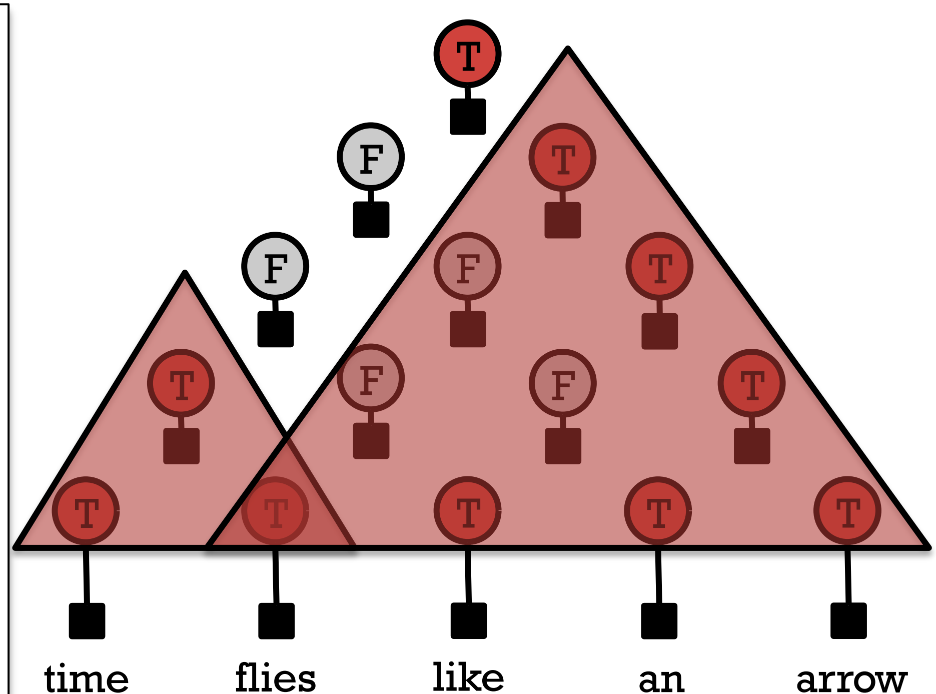
Unlabeled Constituency Parsing

Given: a sentence.

Predict: unlabeled parse.

We could predict whether each span is present **T** or not **F**.

But nothing prevents non-tree structures.



Sending a messsage **to a variable** from its unary factors takes only $O(d*k^d)$ time where $k=2$ and $d=1$.

(Naradowsky, Vieira, & Smith, 2012)

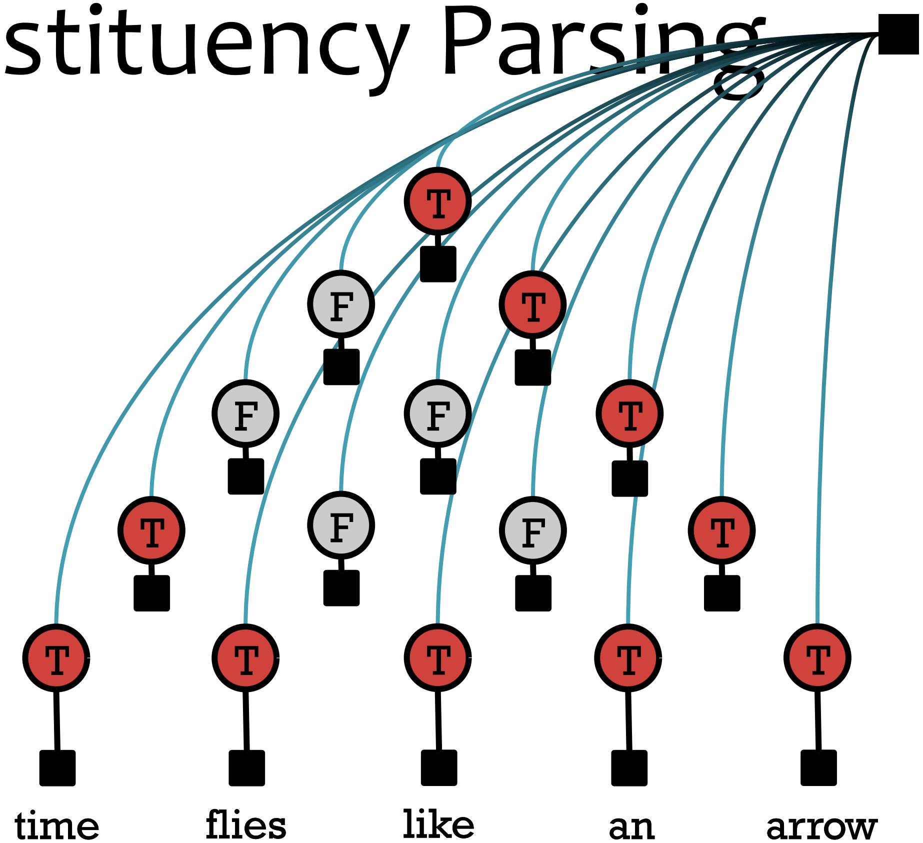
Unlabeled Constituency Parsing

Given: a sentence.

Predict: unlabeled parse.

We could predict whether each span is present T or not F.

But nothing prevents non-tree structures.



Add a *CKYTree* factor which multiplies in *1* if the variables form a tree and *0* otherwise.

(Naradowsky, Vieira, & Smith, 2012)

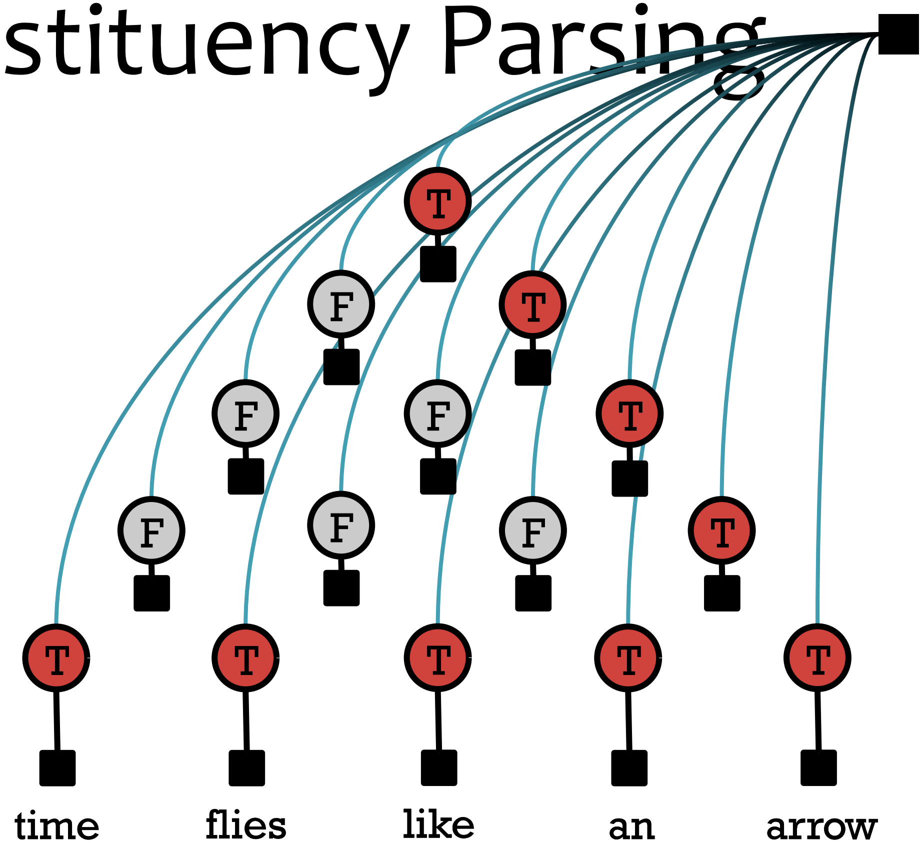
Unlabeled Constituency Parsing

Given: a sentence.

Predict: unlabeled parse.

We could predict whether each span is present T or not F.

But nothing prevents non-tree structures.



Add a *CKYTree* factor which multiplies in *1* if the variables form a tree and *0* otherwise.

(Naradowsky, Vieira, & Smith, 2012)

Unlabeled Constituency Parsing

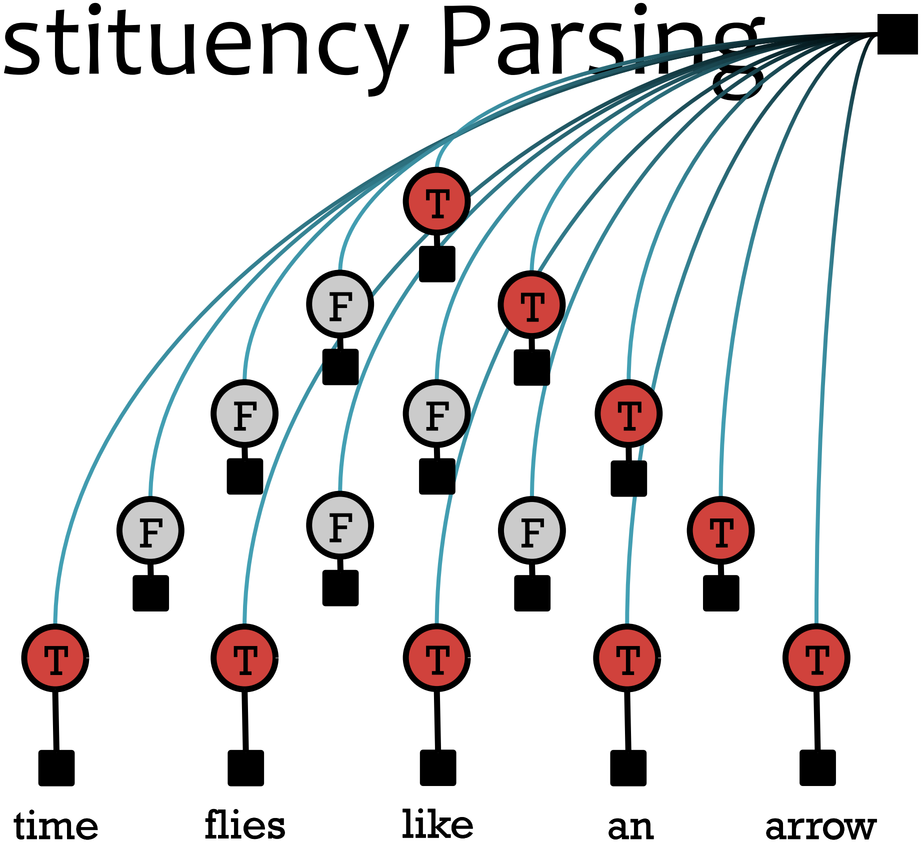
How long does it take to send a message to a variable from the the *CKYTree* factor?

For the given sentence, $O(d*k^d)$ time where $k=2$ and $d=15$.

For a length n sentence, this will be $O(2^{n*n})$.

But we know an algorithm (inside-outside) to compute **all the marginals** in $O(n^3)$…

So can't we do better?



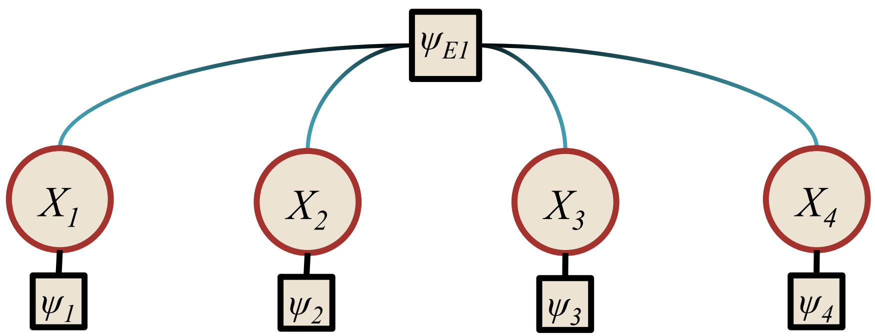
Add a *CKYTree* factor which multiplies in *1* if the variables form a tree and *0* otherwise.

(Naradowsky, Vieira, & Smith, 2012)

Example: The *Exactly1* Factor

Variables: d binary variables $X_1, \ldots, X_d$

Global Factor: $Exactly1(X_1, \ldots, X_d) = \begin{cases} 1 \text{ if exactly one of the } d \text{ binary variables } X_i \text{ is } \textbf{on}, \\ 0 \text{ otherwise} \end{cases}$



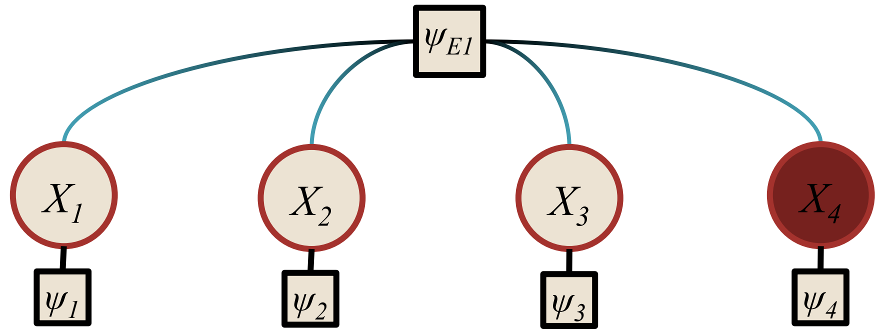
(Smith & Eisner, 2008)

Example: The *Exactly1* Factor

Variables: d binary variables $X_1, \ldots, X_d$

Global Factor: $Exactly1(X_1, \ldots, X_d) =$

- 1 if exactly one of the d binary variables X_i is **on**,
- 0 otherwise

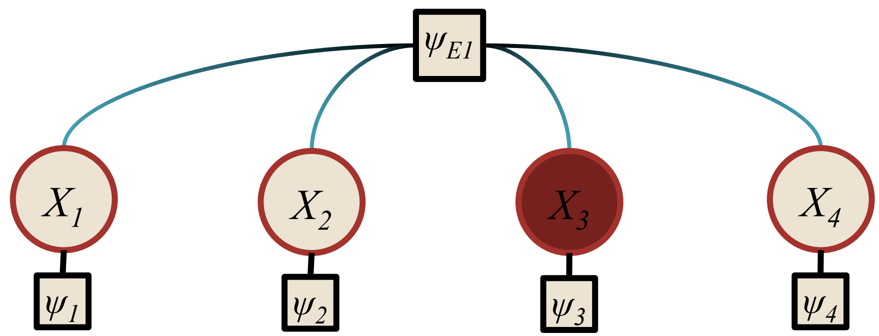


(Smith & Eisner, 2008)

Example: The *Exactly1* Factor

Variables: d binary variables $X_1, \ldots, X_d$

Global Factor: $Exactly1(X_1, \ldots, X_d) = \begin{cases} 1 \text{ if exactly one of the } d \text{ binary variables } X_i \text{ is } \textbf{on}, \\ 0 \text{ otherwise} \end{cases}$

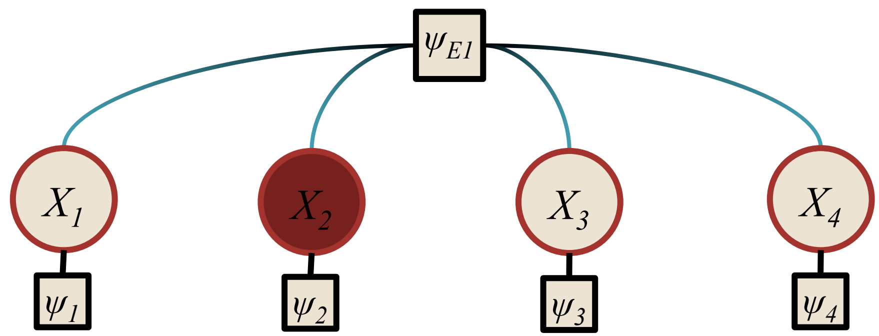


(Smith & Eisner, 2008)

Example: The *Exactly1* Factor

Variables: d binary variables $X_1, \ldots, X_d$

Global Factor: $Exactly1(X_1, \ldots, X_d) = \begin{cases} 1 \text{ if exactly one of the } d \text{ binary variables } X_i \text{ is } \textbf{on}, \\ 0 \text{ otherwise} \end{cases}$

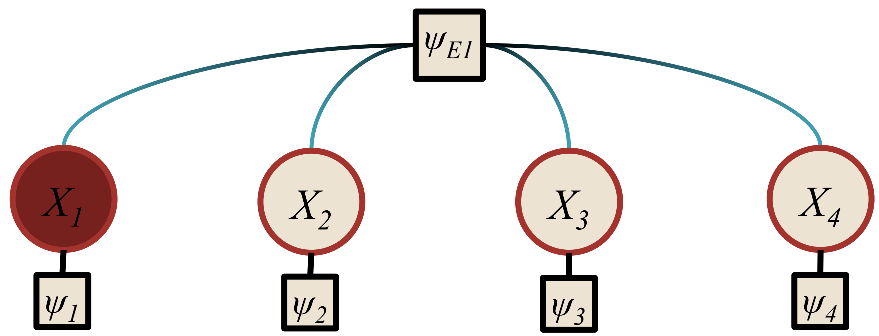


(Smith & Eisner, 2008)

Example: The *Exactly1* Factor

Variables: d binary variables $X_1, \ldots, X_d$

Global Factor: $Exactly1(X_1, \ldots, X_d) = \begin{cases} 1 \text{ if exactly one of the } d \text{ binary variables } X_i \text{ is } \textbf{on}, \\ 0 \text{ otherwise} \end{cases}$

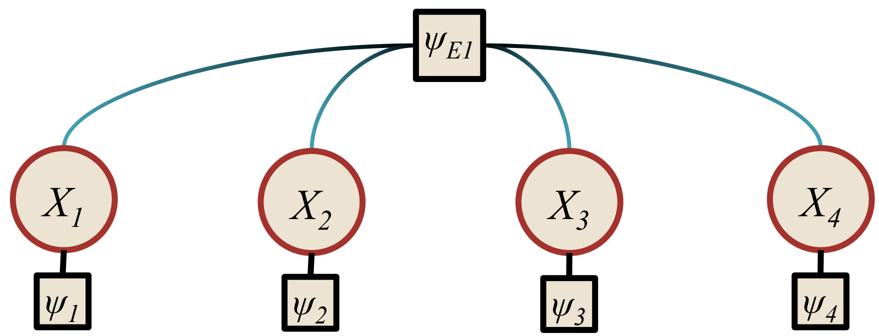


(Smith & Eisner, 2008)

Example: The *Exactly1* Factor

Variables: d binary variables $X_1, \ldots, X_d$

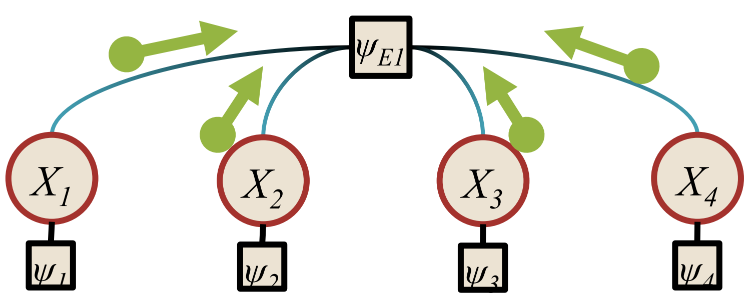
Global Factor: $Exactly1(X_1, \ldots, X_d) = \begin{cases} 1 & \text{if exactly one of the } d \text{ binary variables } X_i \text{ is } \textbf{on}, \\ 0 & \text{otherwise} \end{cases}$



(Smith & Eisner, 2008)

Messages: The *Exactly1* Factor

From Variables

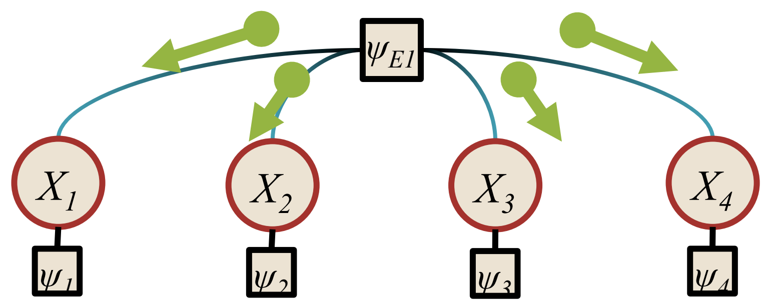


$$\mu_{i \to \alpha}(x_i) = \prod_{\alpha \in \mathcal{N}(i) \setminus \alpha} \mu_{\alpha \to i}(x_i)$$

$O(d*2)$

d = # of neighboring factors

To Variables



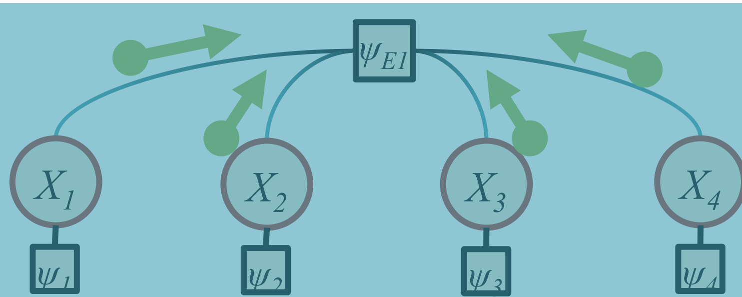
$$\mu_{\alpha \to i}(x_i) = \sum_{\boldsymbol{x}_{\boldsymbol{\alpha}} : \boldsymbol{x}_{\boldsymbol{\alpha}}[i] = x_i} \psi_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}}) \prod_{j \in \mathcal{N}(\alpha) \setminus i} \mu_{j \to \alpha}(\boldsymbol{x}_{\boldsymbol{\alpha}}[i])$$

$O(d*2^d)$

d = # of neighboring variables

Messages: The *Exactly1* Factor

From Variables



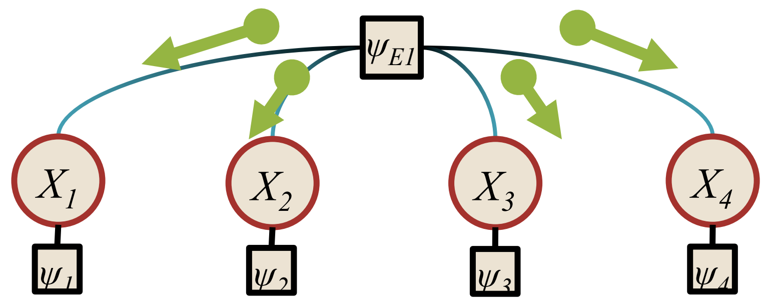
Fast!

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

O(d*2)

d = # of neighboring factors

To Variables



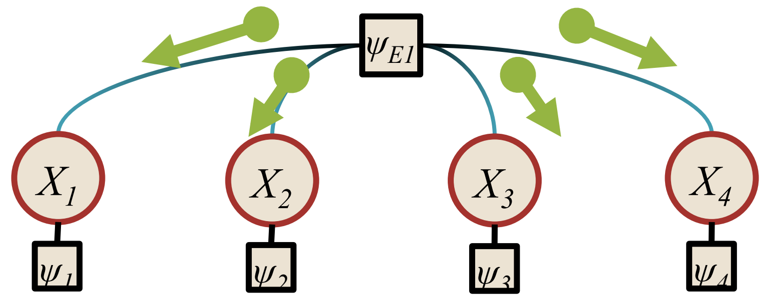
$$\mu_{\alpha\to i}(x_i) = \sum_{\boldsymbol{x}_{\boldsymbol{\alpha}}:\boldsymbol{x}_{\boldsymbol{\alpha}}[i]=x_i} \psi_\alpha(\boldsymbol{x}_{\boldsymbol{\alpha}}) \prod_{j\in\mathcal{N}(\alpha)\setminus i} \mu_{j\to\alpha}(\boldsymbol{x}_{\boldsymbol{\alpha}}[i])$$

O(d*2^d)

d = # of neighboring variables

Messages: The *Exactly1* Factor

To Variables



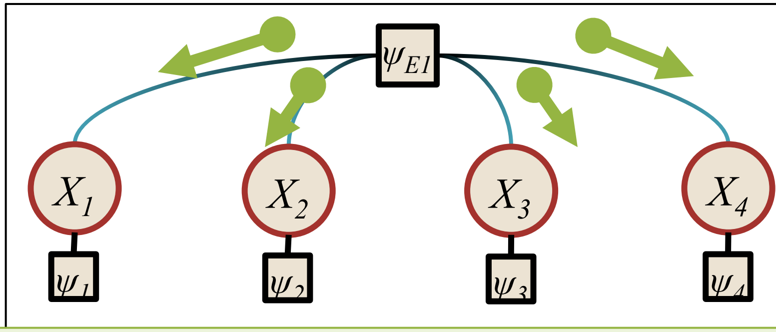
$$\mu_{\alpha \to i}(x_i) = \sum_{\boldsymbol{x_\alpha} : \boldsymbol{x_\alpha}[i] = x_i} \psi_\alpha(\boldsymbol{x_\alpha}) \prod_{j \in \mathcal{N}(\alpha) \setminus i} \mu_{j \to \alpha}(\boldsymbol{x_\alpha}[i])$$

$O(d*2^d)$

d = # of neighboring variables

Messages: The *Exactly1* Factor

To Variables



But the **outgoing** messages from the *Exactly1* factor are defined as a sum over the 2^d possible assignments to $X_1, \ldots, X_d$.

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

Conveniently, $\psi_{E1}(x_a)$ is 0 for all but d values – so **the sum is sparse**!

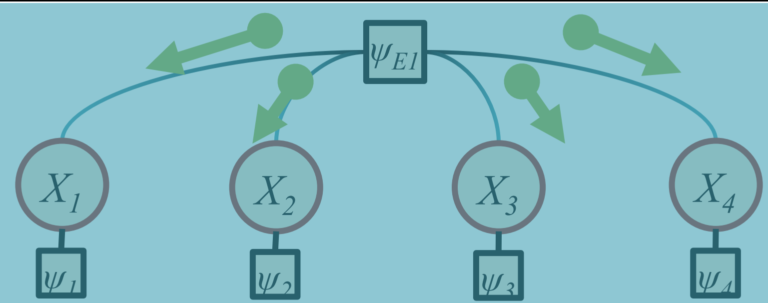
~~$O(d*2^d)$~~

~~d = # of neighboring variables~~

So we can compute all the outgoing messages from ψ_{E1} in $O(d)$ time!

Messages: The *Exactly1* Factor

To Variables



But the **outgoing** messages from the *Exactly1* factor are defined as a sum over the 2^d possible assignments to $X_1, \ldots, X_d$.

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

Fast!

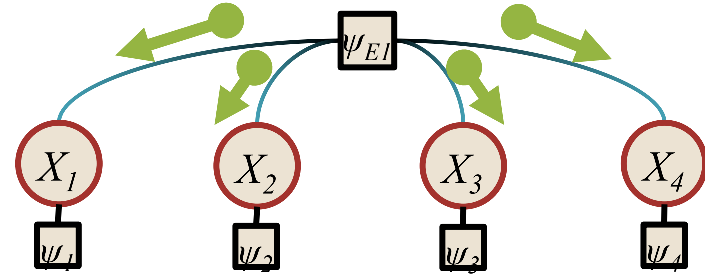
Conveniently, $\psi_{E1}(x_a)$ is 0 for all but d values – so **the sum is sparse**!

$\cancel{O(d*2^d)}$

d = # of neighboring variables

So we can compute all the outgoing messages from ψ_{E1} in $O(d)$ time!

Messages: The *Exactly1* Factor



A factor has a **belief about each of its variables.**

$$b_\alpha(x_i) = \sum_{\boldsymbol{x_\alpha} : \boldsymbol{x_\alpha}[i] = x_i} b_\alpha(\boldsymbol{x_\alpha})$$

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

An outgoing message from a factor is the factor's belief with the **incoming message divided out.**

$$\boxed{\mu_{\alpha \to i}(v) = \frac{b_\alpha(x_i)}{\mu_{i \to \alpha}(v)}}$$

We can compute the Exactly1 factor's beliefs about each of its variables **efficiently.** (Each of the parenthesized terms needs to be computed **only once** for all the variables.)

$$b_\alpha(X_i = 1) = \left(\prod_{j \in \mathcal{N}(\alpha)} \mu_{j \to \alpha}(0) \right) \frac{\mu_{i \to \alpha}(1)}{\mu_{i \to \alpha}(0)}$$

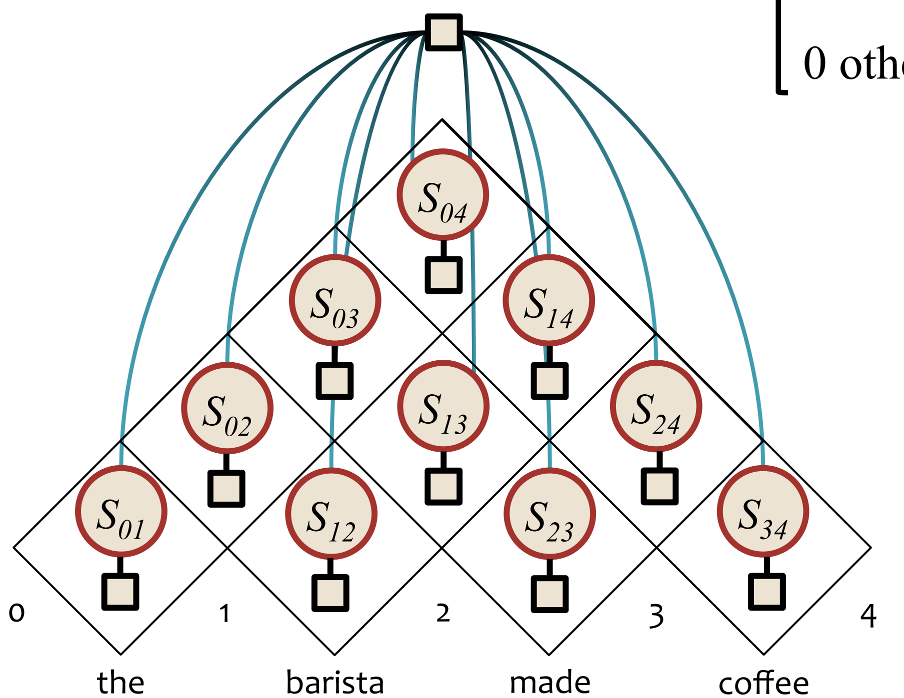
$$b_\alpha(X_i = 0) = \left(\sum_{j=1}^{n} b_\alpha(X_j = 1) \right) - b_\alpha(X_i = 1)$$

(Smith & Eisner, 2008)

Example: The *CKYTree* Factor

Variables: $O(n^2)$ binary variables S_{ij}

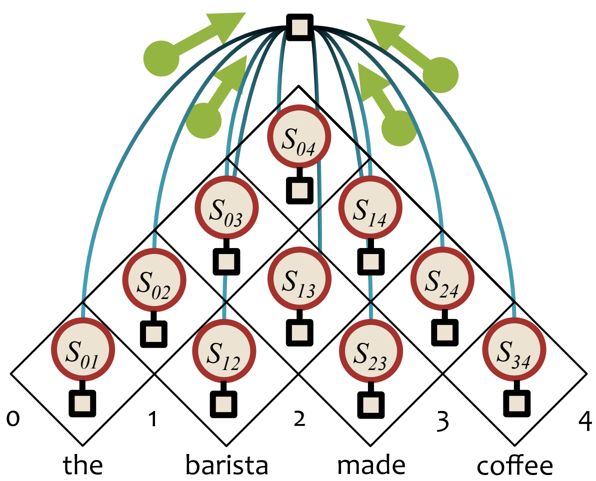
Global Factor: $CKYTree(S_{01}, S_{12}, \ldots, S_{04}) = \begin{cases} 1 \text{ if the span variables form a constituency tree,} \\ 0 \text{ otherwise} \end{cases}$



(Naradowsky, Vieira, & Smith, 2012)

Messages: The *CKYTree* Factor

From Variables

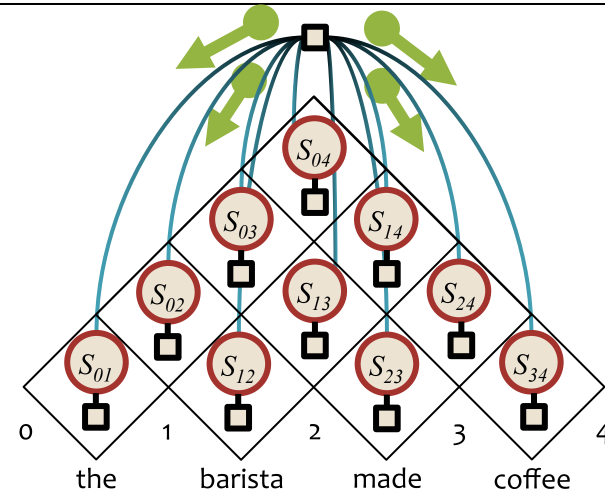


$$\mu_{i\to\alpha}(x_i) = \prod_{\alpha \in \mathcal{N}(i)\setminus\alpha} \mu_{\alpha\to i}(x_i)$$

O(d*2)

d = # of neighboring factors

To Variables



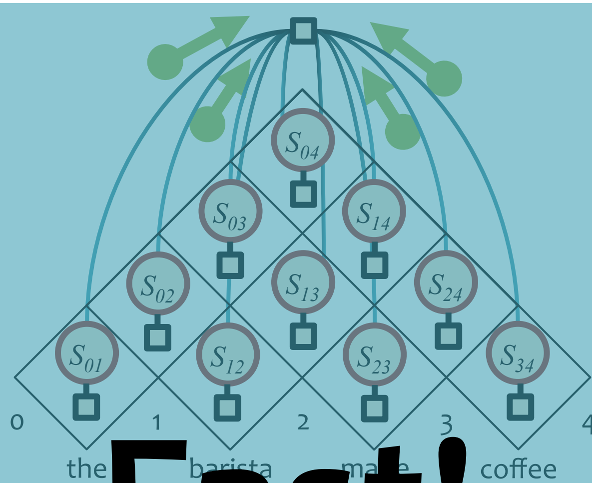
$$\mu_{\alpha\to i}(x_i) = \sum_{\boldsymbol{x}_\alpha : \boldsymbol{x}_\alpha[i] = x_i} \psi_\alpha(\boldsymbol{x}_\alpha) \prod_{j \in \mathcal{N}(\alpha)\setminus i} \mu_{j\to\alpha}(\boldsymbol{x}_\alpha[i])$$

O(d*2^d)

d = # of neighboring variables

Messages: The *CKYTree* Factor

From Variables



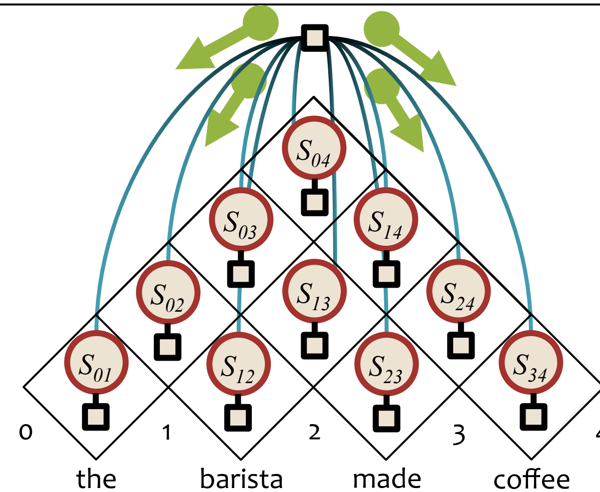
Fast!

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

O(d*2)

d = # of neighboring factors

To Variables



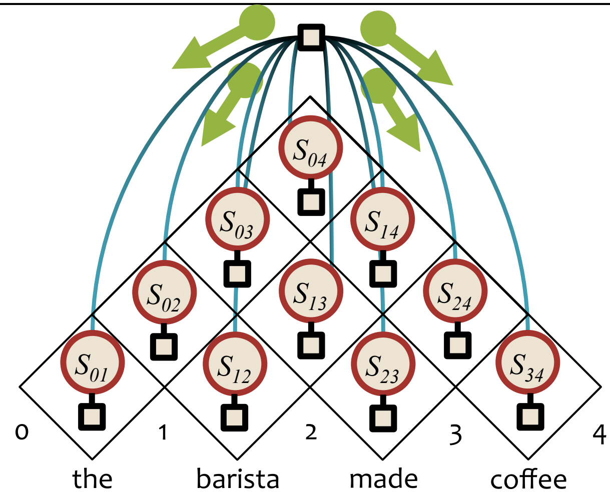
$$\mu_{\alpha\to i}(x_i) = \sum_{\boldsymbol{x_\alpha} : \boldsymbol{x_\alpha}[i] = x_i} \psi_\alpha(\boldsymbol{x_\alpha}) \prod_{j \in \mathcal{N}(\alpha) \setminus i} \mu_{j\to\alpha}(\boldsymbol{x_\alpha}[i])$$

O(d*2^d)

d = # of neighboring variables

Messages: The *CKYTree* Factor

To Variables



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

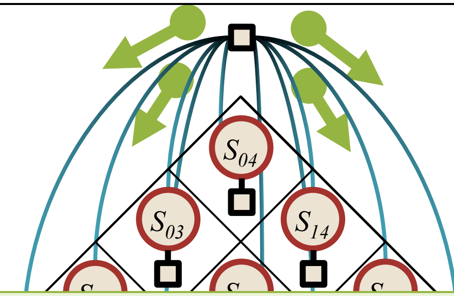
$O(d*2^d)$

d = # of neighboring variables

Messages: The *CKYTree* Factor

But the **outgoing** messages from the *CKYTree* factor are defined as a sum over the $O(2^{n*n})$ possible assignments to $\{S_{ij}\}$.

To Variables



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

$\psi_{CKYTree}(x_a)$ is *1* for exponentially many values in the sum – **but they all correspond to trees!**

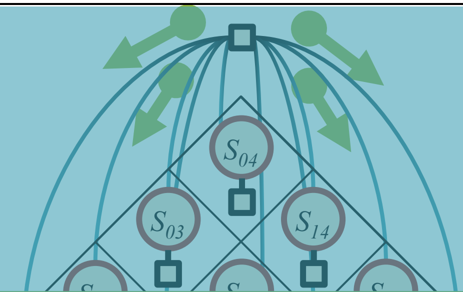
$\cancel{O(d*2^d)}$

~~d = # of neighboring variables~~

With inside-outside we can compute all the outgoing messages from *CKYTree* in $O(n^3)$ time!

Messages: The *CKYTree* Factor

To Variables



But the **outgoing** messages from the *CKYTree* factor are defined as a sum over the $O(2^{n*n})$ possible assignments to $\{S_{ij}\}$.

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

Fast!

$\psi_{CKYTree}(x_a)$ is 1 for exponentially many values in the sum – **but they all correspond to trees!**

$\cancel{O(d*2^d)}$

~~d = # of neighboring variables~~

With inside-outside we can compute all the outgoing messages from *CKYTree* in $O(n^3)$ time!

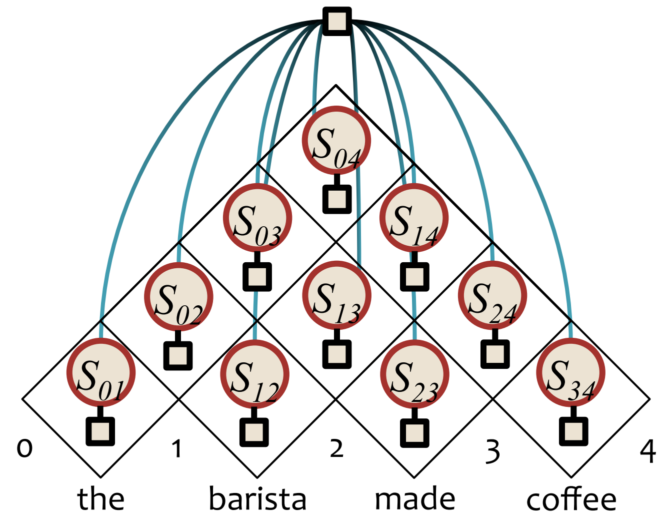
Example: The *CKYTree* Factor

For a length n sentence, define an anchored weighted context free grammar (WCFG).

Each span is weighted by the ratio of the incoming messages from the corresponding span variable.

$$w({}_iX_j \rightarrow {}_iX_k \; {}_kX_j) = \frac{\mu_{S_{ij}\rightarrow\psi}(1)}{\mu_{S_{ij}\rightarrow\psi}(0)}$$

$$w({}_iX_{i+1} \rightarrow a_{i+1}) = \frac{\mu_{S_{i,i+1}\rightarrow\psi}(1)}{\mu_{S_{i,i+1}\rightarrow\psi}(0)}$$



Run the inside-outside algorithm on the sentence a_1, a_1, …, a_n with the anchored WCFG.

$$\mu_{S_{ij}\rightarrow\psi}(1) = \frac{\text{outside}({}_iX_j)}{\text{inside}({}_0X_n)}$$

$$\mu_{S_{ij}\rightarrow\psi}(0) = 1 - w({}_iX_j \rightarrow {}_iX_k \; {}_kX_j)\frac{\text{outside}({}_iX_j)}{\text{inside}({}_0X_n)}$$

(Naradowsky, Vieira, & Smith, 2012)

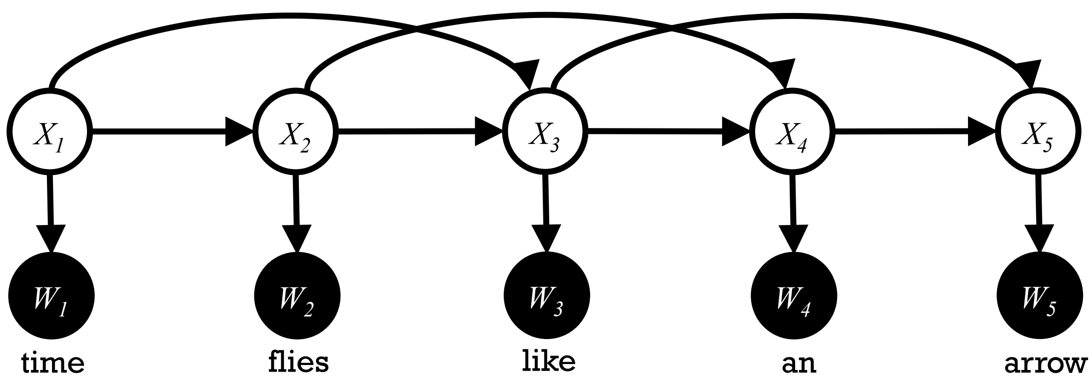
JOHNS HOPKINS UNIVERSITY

Example: The *TrigramHMM* Factor

Factors can compactly encode the preferences of an entire sub-model.

Consider the joint distribution of a trigram HMM over 5 variables:

- It's traditionally defined as a Bayes Network



(Smith & Eisner, 2008)

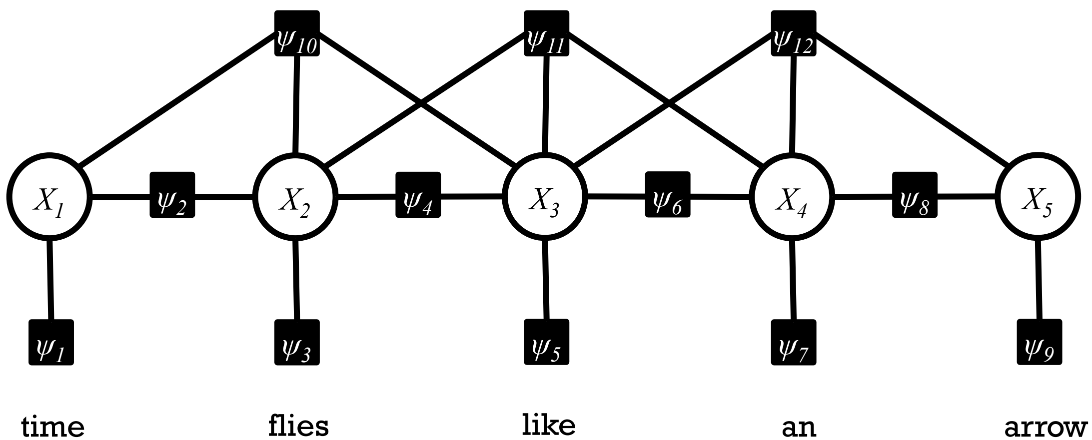
Example: The *TrigramHMM* Factor

Factors can compactly encode the preferences of an entire sub-model.

Consider the joint distribution of a trigram HMM over 5 variables:

– It's traditionally defined as a Bayes Network

– But we can represent it as a (loopy) factor graph



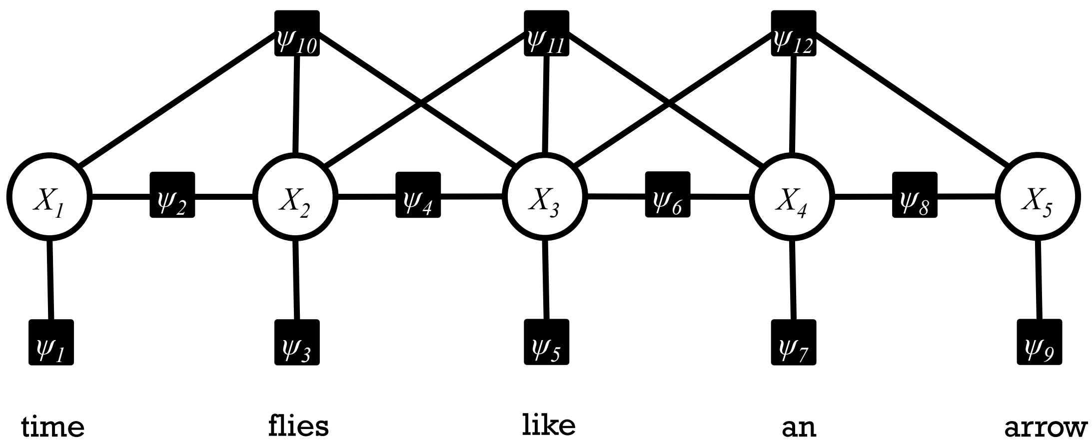
(Smith & Eisner, 2008)

Example: The *TrigramHMM* Factor

Factors can compactly encode the preferences of an entire sub-model.

Consider the joint distribution of a trigram HMM over 5 variables:

- It's traditionally defined as a Bayes Network
- But we can represent it as a (loopy) factor graph
- We could even pack all those factors into a single *TrigramHMM* factor



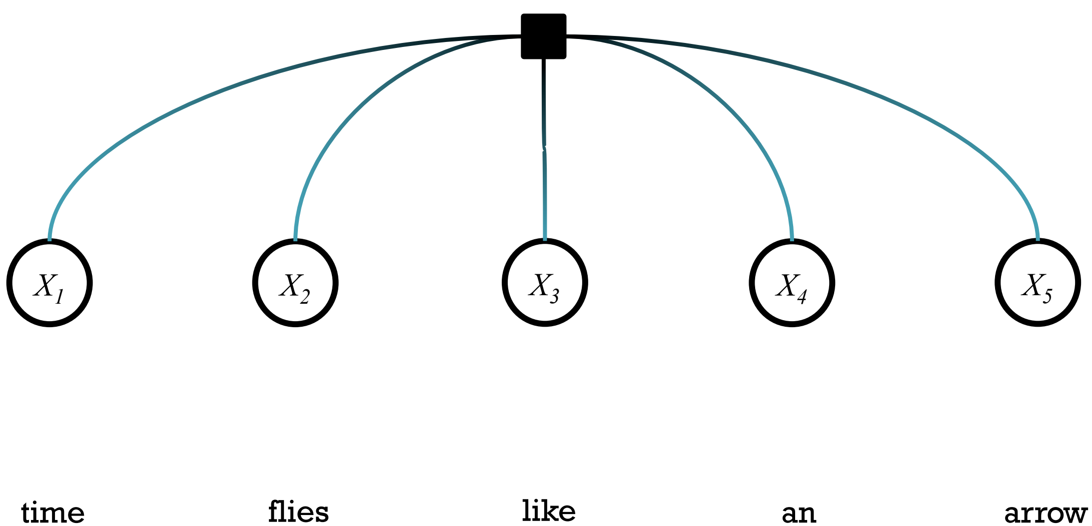
(Smith & Eisner, 2008)

Example: The *TrigramHMM* Factor

Factors can compactly encode the preferences of an entire sub-model.

Consider the joint distribution of a trigram HMM over 5 variables:

- It's traditionally defined as a Bayes Network
- But we can represent it as a (loopy) factor graph
- We could even pack all those factors into a single *TrigramHMM* factor

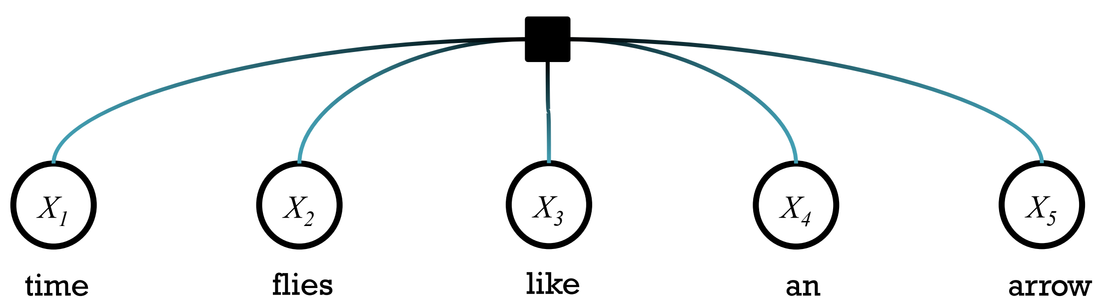


(Smith & Eisner, 2008)

Example: The *TrigramHMM* Factor

Variables: d discrete variables $X_1, \ldots, X_d$

Global Factor: $TrigramHMM\,(X_1, \ldots, X_d) = p(X_1, \ldots, X_d)$ according to a trigram HMM model



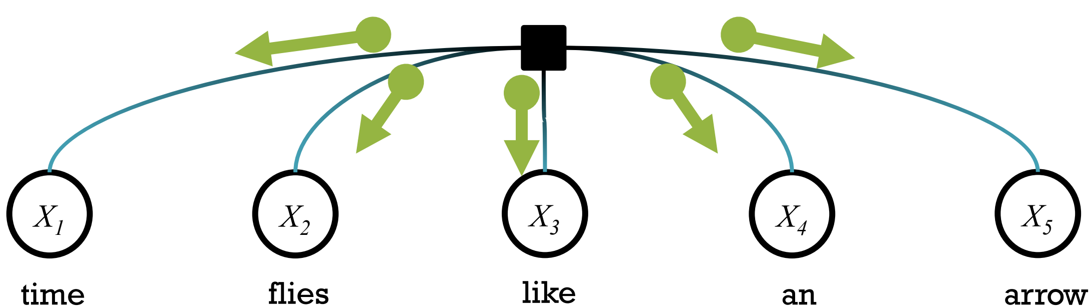
(Smith & Eisner, 2008)

Example: The *TrigramHMM* Factor

Variables: d discrete variables $X_1, \ldots, X_d$

Global Factor: $TrigramHMM\,(X_1, \ldots, X_d) = p(X_1, \ldots, X_d)$ according to a trigram HMM model

Compute outgoing messages **efficiently** with the standard trigram HMM dynamic programming algorithm (junction tree)!



(Smith & Eisner, 2008)

Combinatorial Factors

- Usually, it takes $O(k^d)$ time to compute outgoing messages from a factor over d variables with k possible values each.
- But not always:
  1. Factors like **Exactly1** with only polynomially many nonzeroes in the potential table
  2. Factors like **CKYTree** with exponentially many nonzeroes but in a special pattern
  3. Factors like **TrigramHMM** with all nonzeroes but which factor further

Combinatorial Factors

Factor graphs can encode structural constraints on many variables via constraint factors.

Example NLP constraint factors:

– Projective and non-projective **dependency parse** constraint (Smith & Eisner, 2008)
– **CCG parse** constraint (Auli & Lopez, 2011)
– Labeled and unlabeled **constituency parse** constraint (Naradowsky, Vieira, & Smith, 2012)
– **Inversion transduction grammar** (ITG) constraint (Burkett & Klein, 2012)

Combinatorial Optimization within Max-Product

- **Max-product BP** computes **max-marginals.**
  - The max-marginal $b_i(x_i)$ is the (unnormalized) probability of the MAP assignment under the constraint $X_i = x_i$.
- Duchi et al. (2006) define factors, over many variables, for which efficient combinatorial optimization algorithms exist.
  - **Bipartite matching:** max-marginals can be computed with standard max-flow algorithm and the Floyd-Warshall all-pairs shortest-paths algorithm.
  - **Minimum cuts:** max-marginals can be computed with a min-cut algorithm.
- Similar to sum-product case: the combinatorial algorithms are **embedded** within the standard loopy BP algorithm.

(Duchi, Tarlow, Elidan, & Koller, 2006)

Structured BP vs. Dual Decomposition

	Sum-product BP	Max-product BP	Dual Decomposition
Output	Approximate marginals	Approximate MAP assignment	True MAP assignment (with branch-and-bound)
Structured Variant	Coordinates marginal inference algorithms	Coordinates MAP inference algorithms	Coordinates MAP inference algorithms
Example Embedded Algorithms	- Inside-outside - Forward-backward	- CKY - Viterbi algorithm	- CKY - Viterbi algorithm

(Koo et al., 2010; Rush et al., 2010)

(Duchi, Tarlow, Elidan, & Koller, 2006)

Johns Hopkins University

Additional Resources

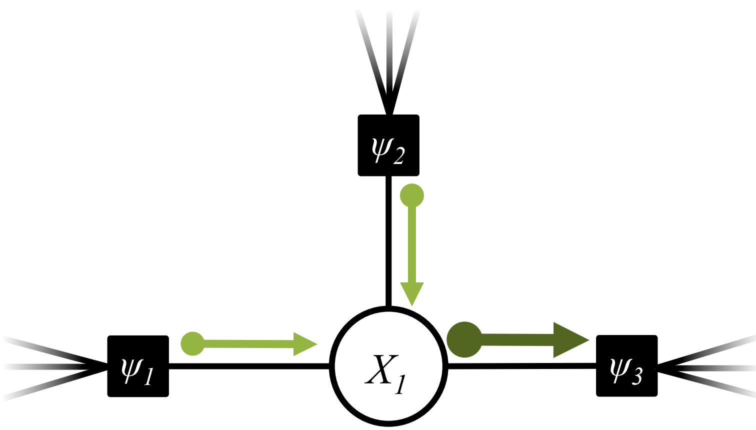
See **NAACL 2012 / ACL 2013 tutorial** by Burkett & Klein "Variational Inference in Structured NLP Models" for…

- An alternative approach to efficient marginal inference for NLP: **Structured Mean Field**
- Also, includes **Structured BP**

http://nlp.cs.berkeley.edu/tutorials/variational-tutorial-slides.pdf

Sending Messages: Computational Complexity

From Variables



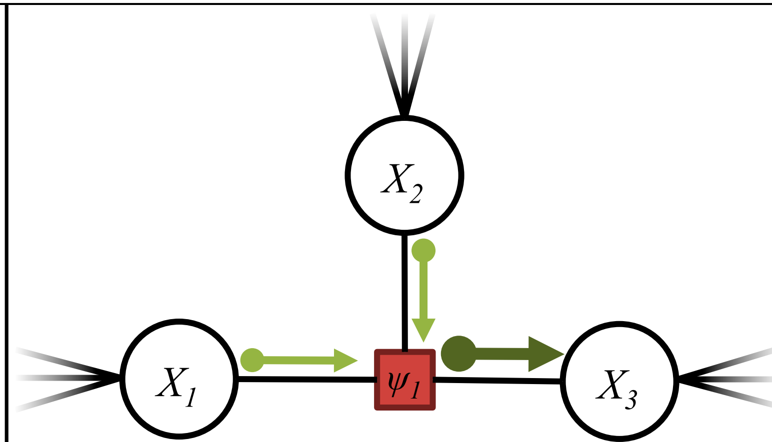
$$\mu_{i\to\alpha}(x_i) = \prod_{\alpha\in\mathcal{N}(i)\setminus\alpha} \mu_{\alpha\to i}(x_i)$$

$O(d^*k)$

d = # of neighboring factors

k = # possible values for X_i

To Variables



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

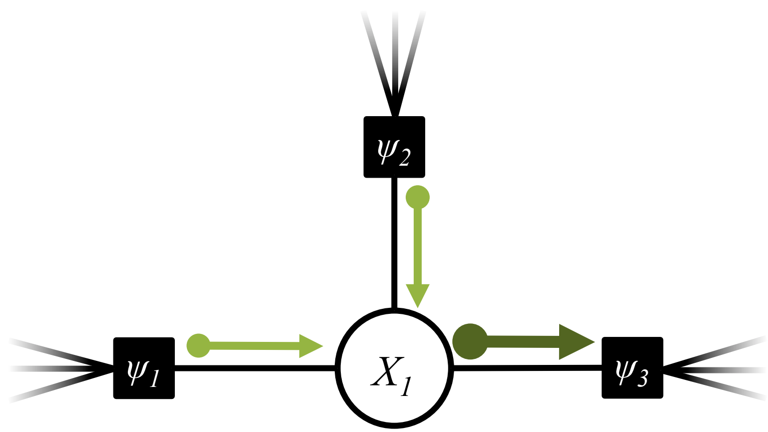
$O(d^*k^d)$

d = # of neighboring variables

k = maximum # possible values for a neighboring variable

Sending Messages: Computational Complexity

From Variables



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

$O(d*k)$

d = # of neighboring factors

k = # possible values for X_i

To Variables

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

$O(d*k^d)$

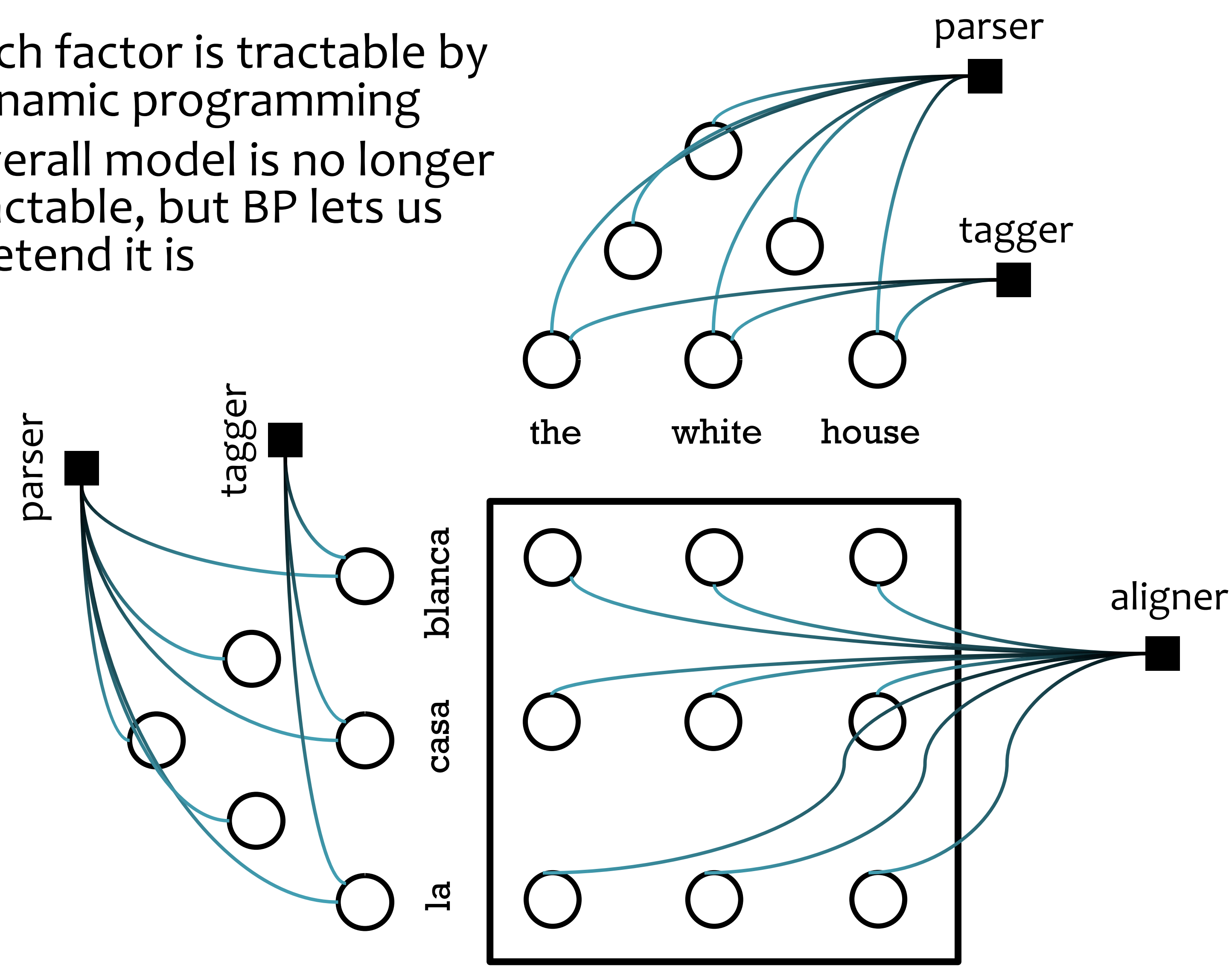
d = # of neighboring variables

k = maximum # possible values for a neighboring variable

INCORPORATING STRUCTURE INTO VARIABLES

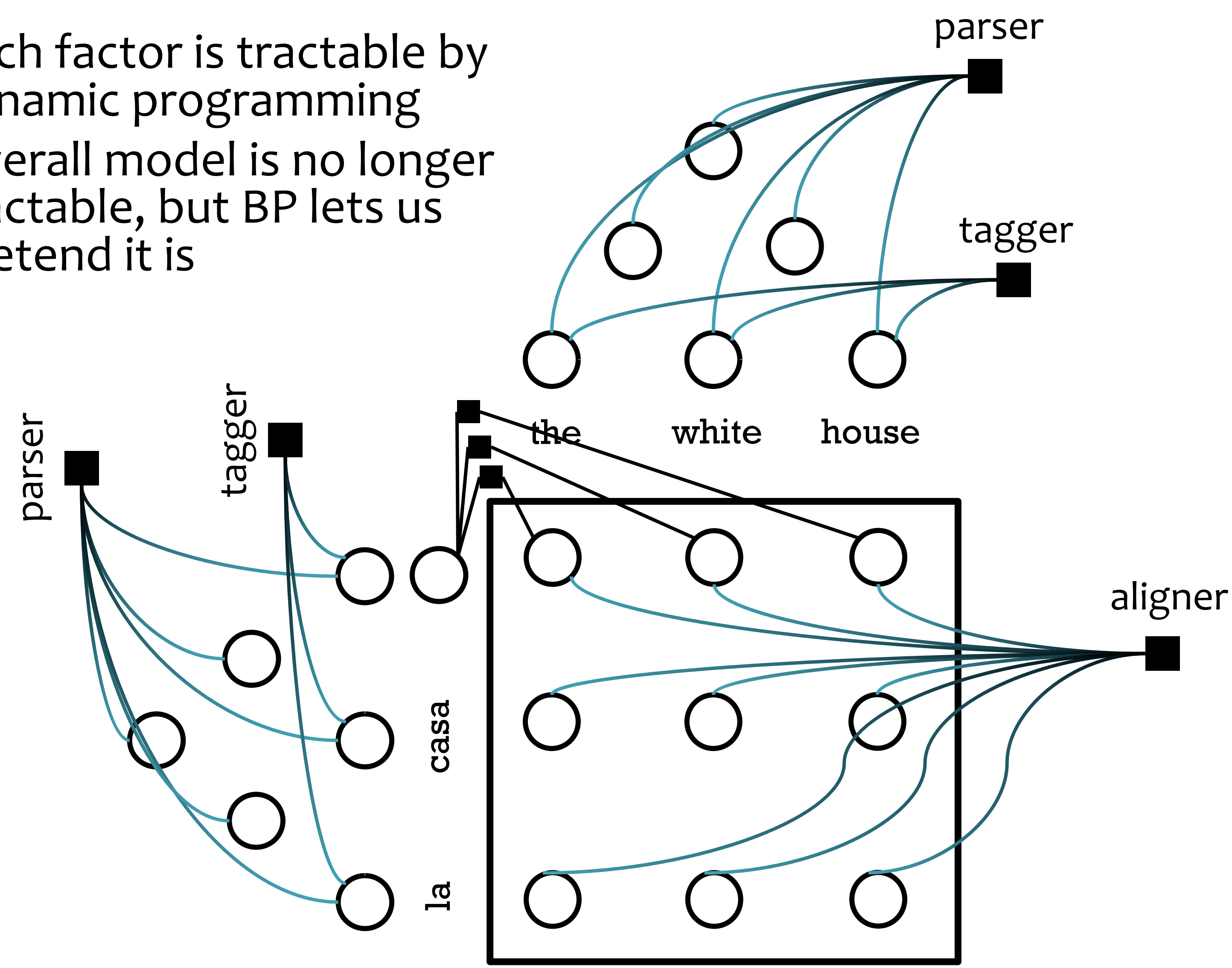
BP for Coordination of Algorithms

- Each factor is tractable by dynamic programming
- Overall model is no longer tractable, but BP lets us pretend it is



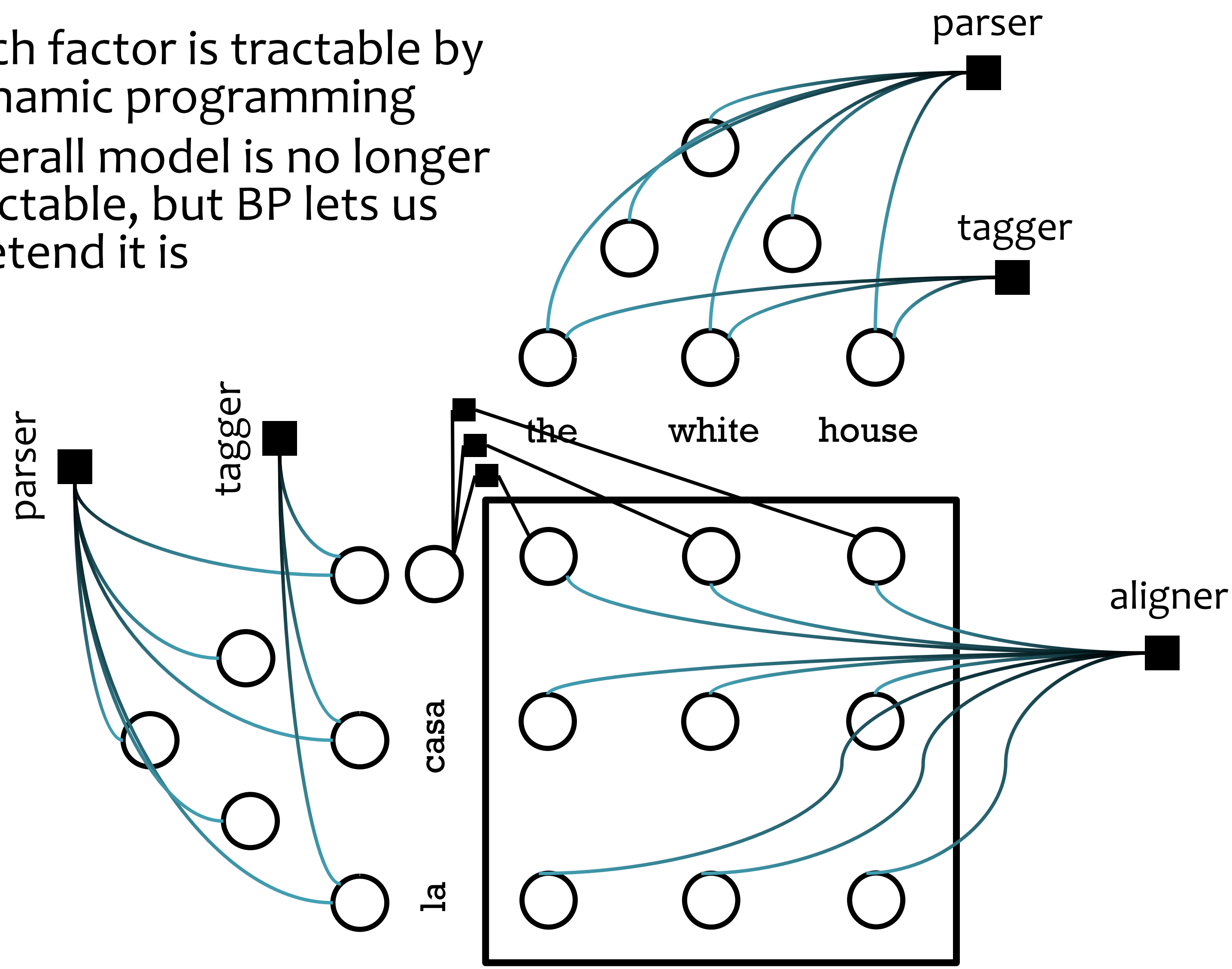
BP for Coordination of Algorithms

- Each factor is tractable by dynamic programming
- Overall model is no longer tractable, but BP lets us pretend it is



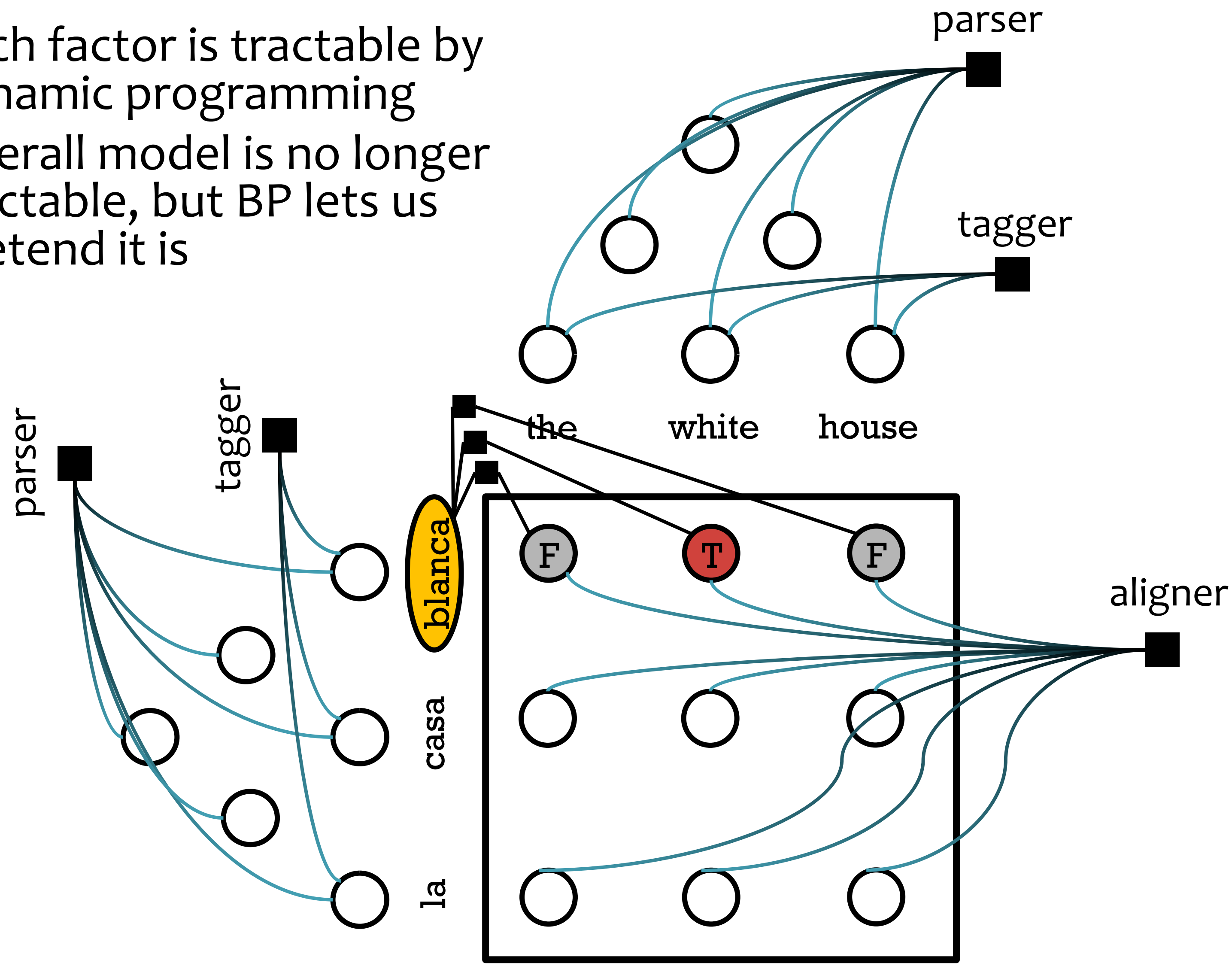
BP for Coordination of Algorithms

- Each factor is tractable by dynamic programming
- Overall model is no longer tractable, but BP lets us pretend it is



BP for Coordination of Algorithms

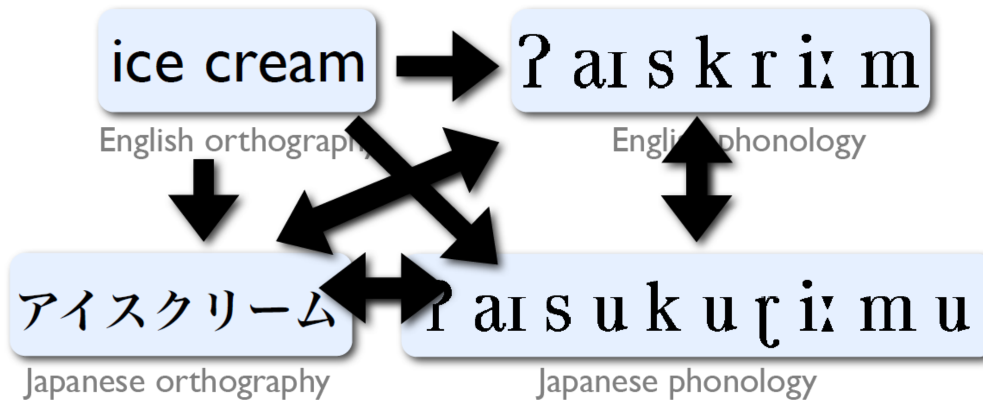
- Each factor is tractable by dynamic programming
- Overall model is no longer tractable, but BP lets us pretend it is



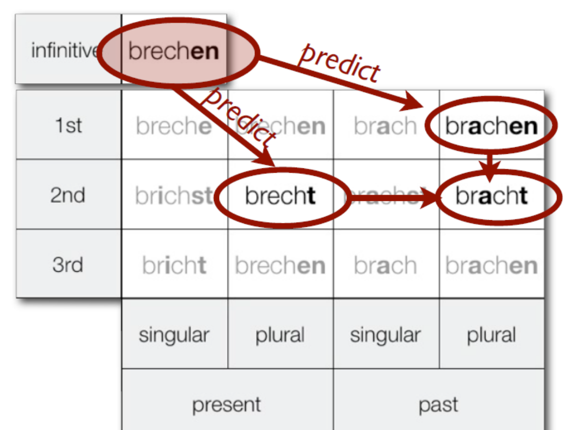
String-Valued Variables

Consider two examples from Section 1:

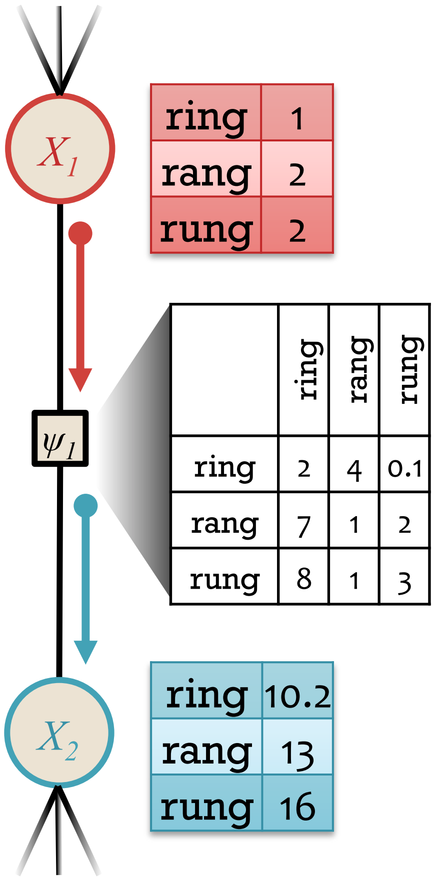
- **Variables (string):**
  - English and Japanese orthographic strings
  - English and Japanese phonological strings
- **Interactions:**
  - All pairs of strings could be relevant



- **Variables (string):**
  - Inflected forms of the same verb
- **Interactions:**
  - Between pairs of entries in the table (e.g. infinitive form affects present-singular)



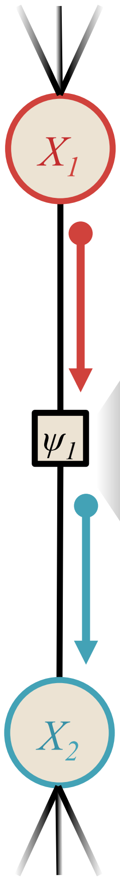
Graphical Models over Strings



- Most of our problems so far:
  - Used **discrete** variables
  - Over a small finite set of **string values**
  - Examples:
    - POS tagging
    - Labeled constituency parsing
    - Dependency parsing
- We use **tensors** (e.g. vectors, matrices) to represent the messages and factors

(Dreyer & Eisner, 2009)

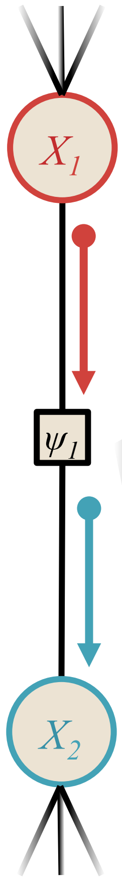
Graphical Models over Strings



ring	1
rang	2
rung	2

	ring	rang	rung
ring	2	4	0.1
rang	7	1	2
rung	8	1	3

ring	10.2
rang	13
rung	16



abacus	0.1
...	...
rang	3
ring	4
rung	5
...	...
zymurgy	0.1

	abacus	...	rang	ring	rung	...	zymurgy
abacus	0.1		0.2	0.1	0.1		0.1
...							
rang	0.1		2	4	0.1		0.1
ring	0.1		7	1	2		0.1
rung	0.2		8	1	3		0.1
...							
zymurgy	0.1		0.1	0.2	0.2		0.1

Time Complexity:

var. → fac. $O(d*k^d)$

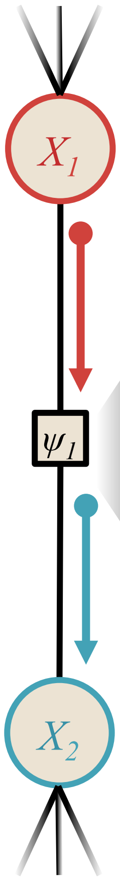
fac. → var. $O(d*k)$

What happens as the # of possible values for a variable, k, increases?

We can still keep the computational complexity down by including only low arity factors (i.e. small d).

(Dreyer & Eisner, 2009)

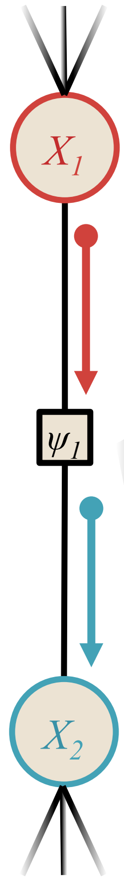
Graphical Models over Strings



ring	1
rang	2
rung	2

	ring	rang	rung
ring	2	4	0.1
rang	7	1	2
rung	8	1	3

ring	10.2
rang	13
rung	16



abacus	0.1
…	…
rang	3
ring	4
rung	5
…	…

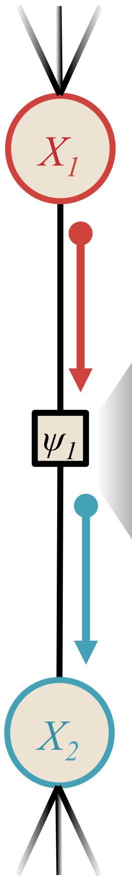
	abacus	…	rang	ring	rung	…
abacus	0.1		0.2	0.1	0.1	
…						
rang	0.1		2	4	0.1	
ring	0.1		7	1	2	
rung	0.2		8	1	3	
…						

But what if the domain of a variable is Σ^*, the **infinite set of all possible strings**?

How can we represent a distribution over **one or more** infinite sets?

(Dreyer & Eisner, 2009)

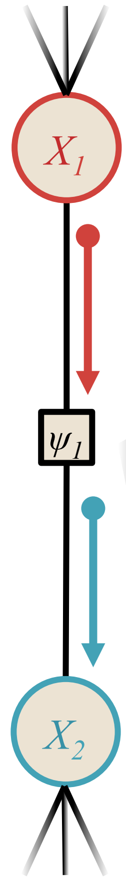
Graphical Models over Strings



ring	1
rang	2
rung	2

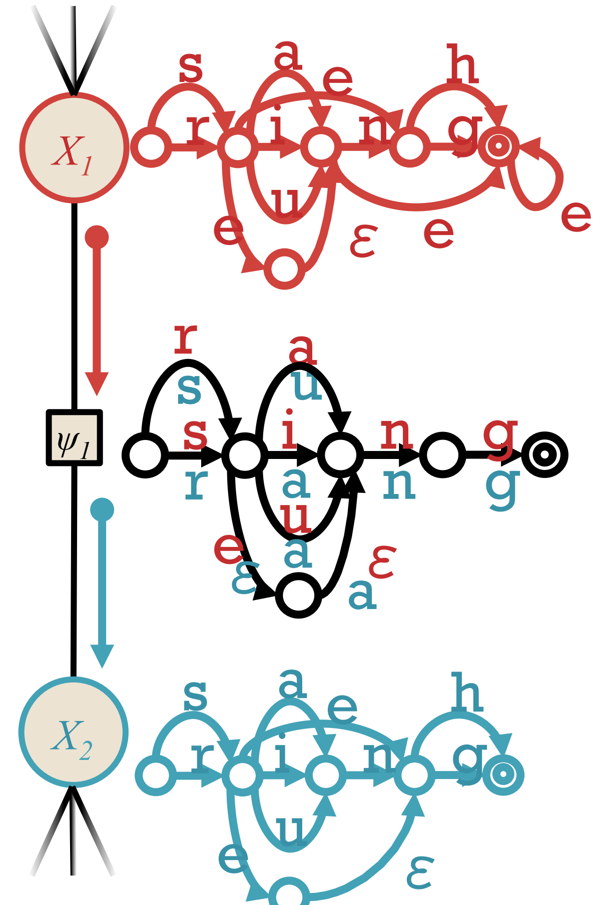
	ring	rang	rung
ring	2	4	0.1
rang	7	1	2
rung	8	1	3

ring	10.2
rang	13
rung	16



abacus	0.1
...	...
rang	3
ring	4
rung	5
...	...

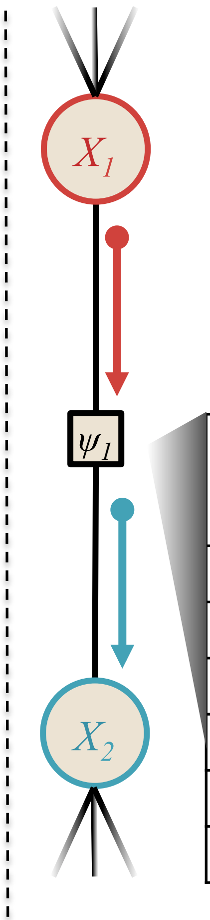
	abacus	...	rang	ring	rung	...
abacus	0.1		0.2	0.1	0.1	
...						
rang	0.1		2	4	0.1	
ring	0.1		7	1	2	
rung	0.2		8	1	3	
...						



Finite State Machines let us represent something infinite in finite space!

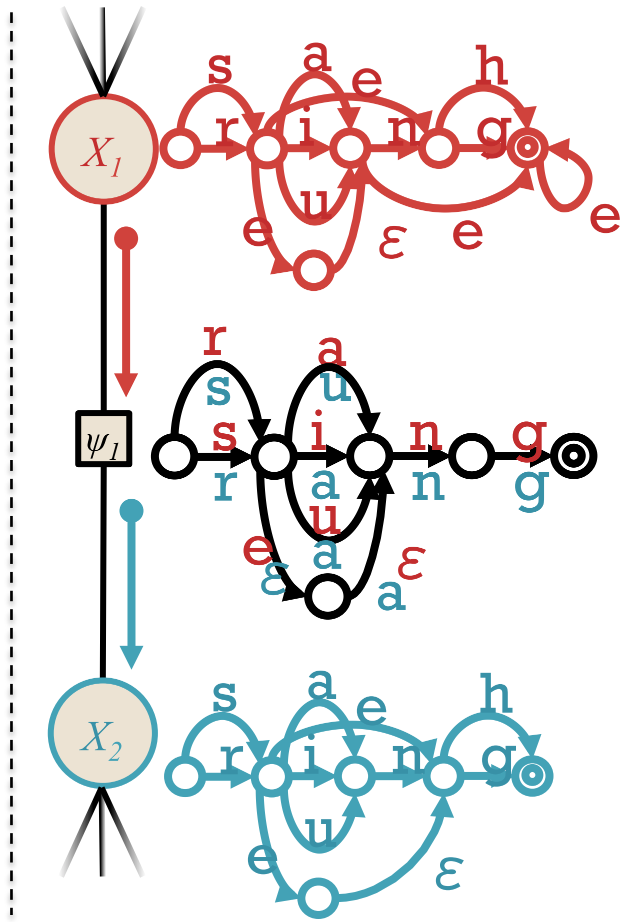
(Dreyer & Eisner, 2009)

Graphical Models over Strings



abacus	0.1
…	…
rang	3
ring	4
rung	5
…	…

	abacus	…	rang	ring	rung	…
abacus	0.1		0.2	0.1	0.1	
…						
rang	0.1		2	4	0.1	
ring	0.1		7	1	2	
rung	0.2		8	1	3	
…						



messages and **beliefs** are Weighted Finite State Acceptors (WFSA)

factors are Weighted Finite State Transducers (WFST)

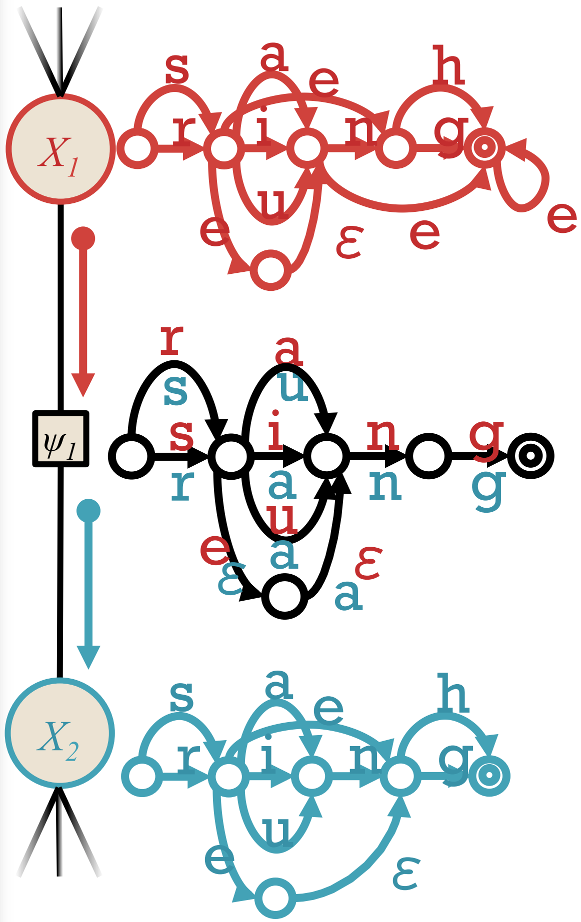
Finite State Machines let us represent something infinite in finite space!

(Dreyer & Eisner, 2009)

Graphical Models over Strings

That solves the problem of **representation.**

But how do we manage the problem of **computation?** (We still need to compute messages and beliefs.)



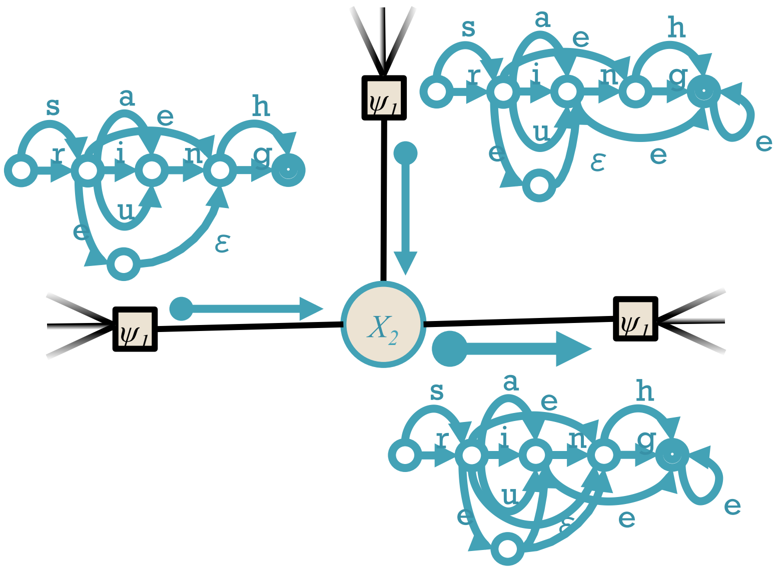
messages and **beliefs** are Weighted Finite State Acceptors (WFSA)

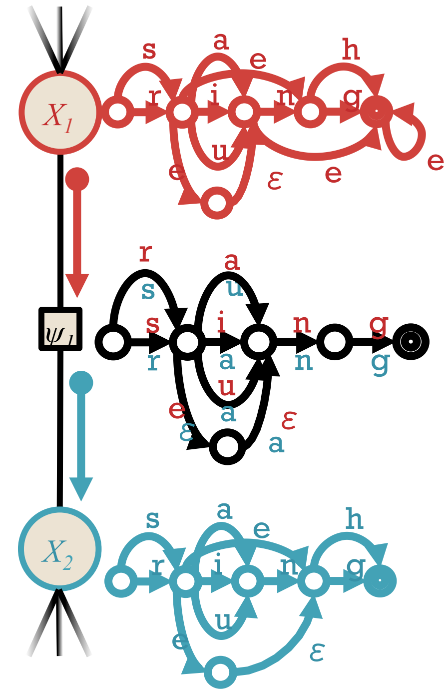
factors are Weighted Finite State Transducers (WFST)

Finite State Machines let us represent something infinite in finite space!

(Dreyer & Eisner, 2009)

Graphical Models over Strings





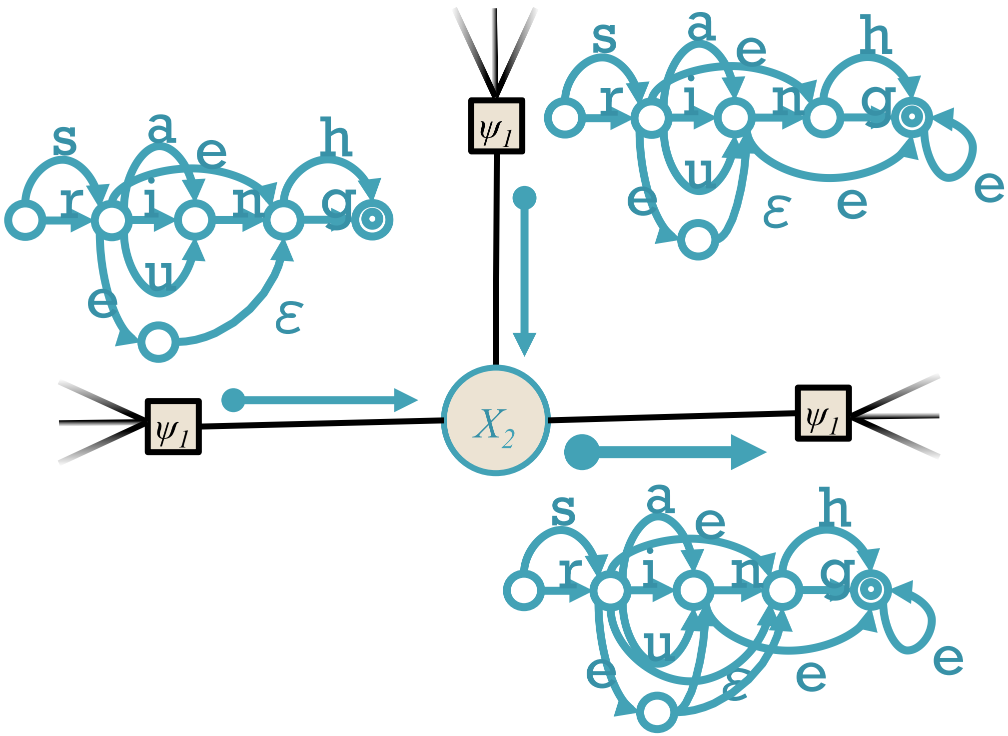
$$\mu_{i \to \alpha}(x_i) = \prod_{\alpha \in \mathcal{N}(i) \setminus \alpha} \mu_{\alpha \to i}(x_i)$$

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

All the message and belief computations simply reuse standard FSM dynamic programming algorithms.

(Dreyer & Eisner, 2009)

Graphical Models over Strings



The pointwise product of two WFSAs is…

…their intersection.

Compute the product of (possibly many) messages $\mu_{\alpha \to i}$ (each of which is a WSFA) via WFSA intersection

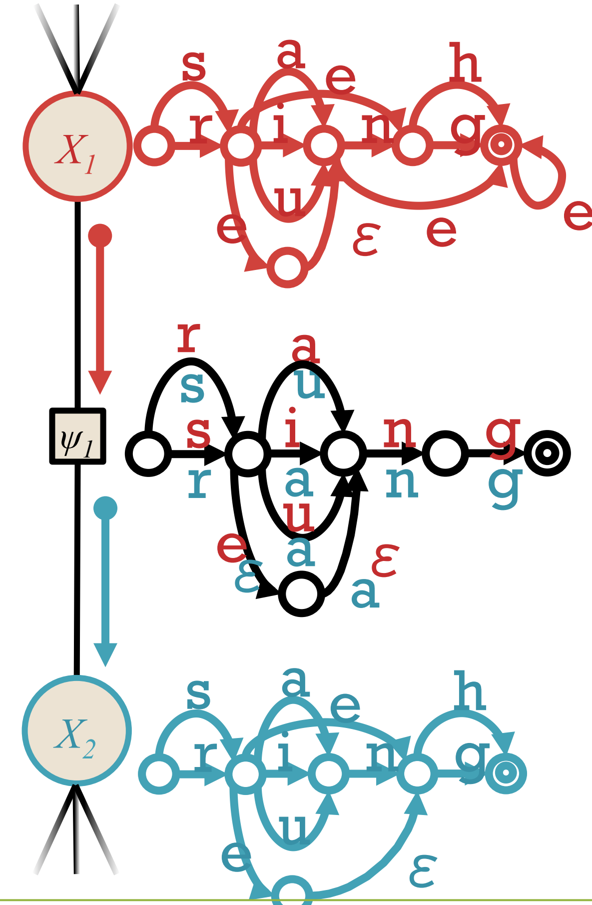
$$\mu_{i \to \alpha}(x_i) = \prod_{\alpha \in \mathcal{N}(i) \setminus \alpha} \mu_{\alpha \to i}(x_i)$$

(Dreyer & Eisner, 2009)

Graphical Models over Strings

Compute marginalized product of WFSA message $\mu_{k\rightarrow\alpha}$ and WFST factor ψ_α, with: `domain(compose(`ψ_α, $\mu_{k\rightarrow\alpha}$`))`

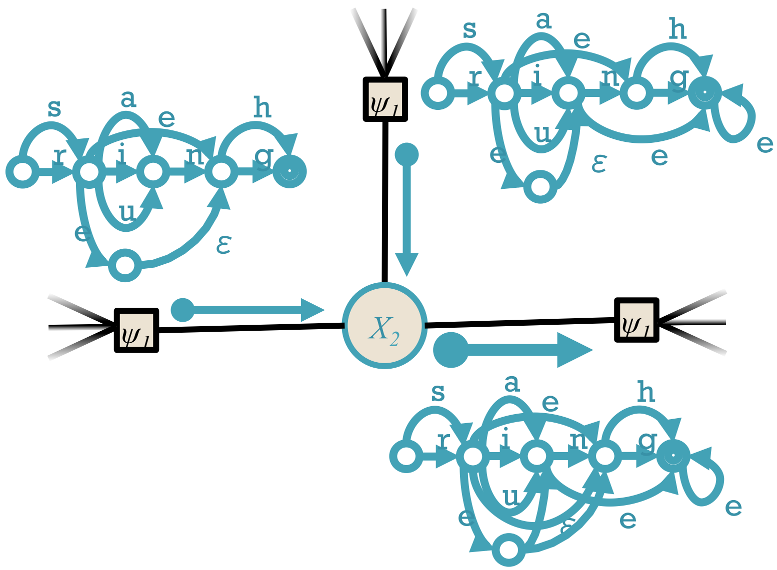
- `compose`: produces a new WFST with a distribution over (X_i, X_j)
- `domain`: marginalizes over X_j to obtain a WFSA over X_i only

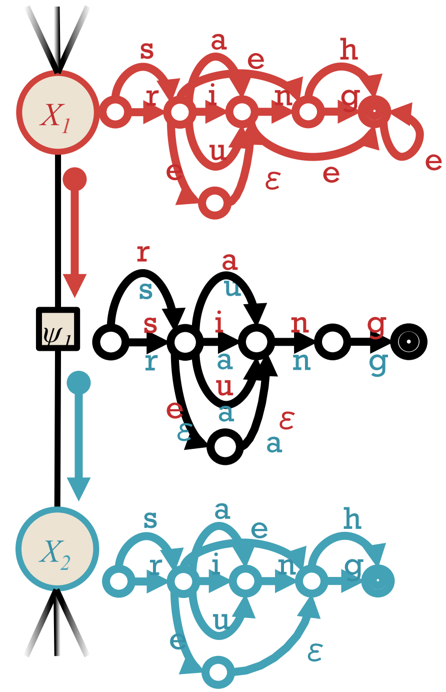


$$\mu_{\alpha\rightarrow i}(x_i) = \sum_{\boldsymbol{x}_\alpha : \boldsymbol{x}_\alpha[i]=x_i} \psi_\alpha(\boldsymbol{x}_\alpha) \prod_{j\in\mathcal{N}(\alpha)\setminus i} \mu_{j\rightarrow\alpha}(\boldsymbol{x}_\alpha[i])$$

(Dreyer & Eisner, 2009)

Graphical Models over Strings





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

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

All the message and belief computations simply reuse standard FSM dynamic programming algorithms.

(Dreyer & Eisner, 2009)

The usual NLP toolbox

- **WFSA:** weighted finite state **automata**
- **WFST:** weighted finite state **transducer**
- ***k*-tape WFSM:** weighted finite state machine jointly mapping between k strings

They each assign a score to a set of strings.

We can interpret them as factors in a graphical model.

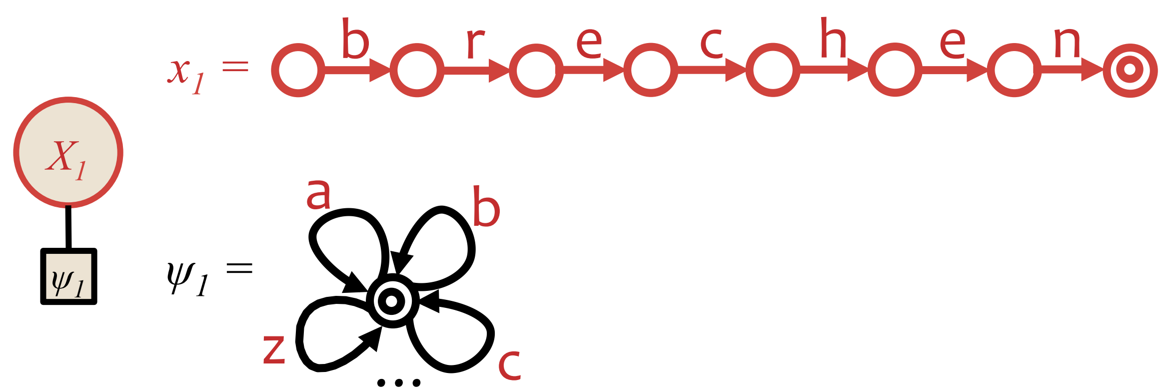
The only difference is the **arity** of the factor.

WFSA as a Factor Graph

- **WFSA:** weighted finite state **automata**
- **WFST:** weighted finite state **transducer**
- ***k*-tape WFSM:** weighted finite state machine jointly mapping between k strings

$\psi_1(x_1) = 4.25$

A **WFSA** is a function which maps a string to a score.

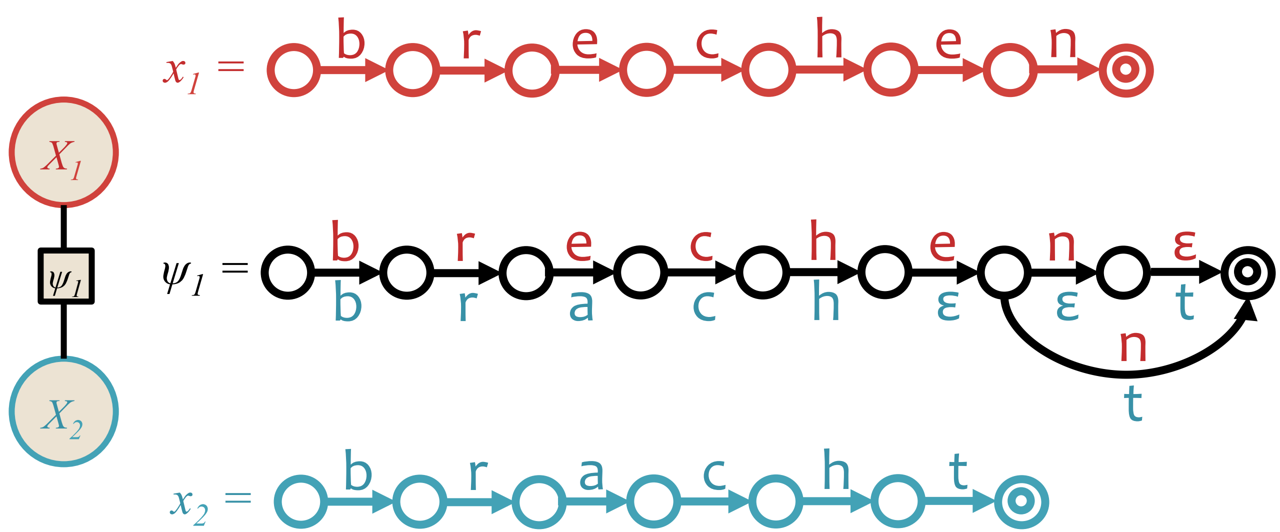


WFST as a Factor Graph

- **WFSA:** weighted finite state **automata**
- **WFST:** weighted finite state **transducer**
- ***k*-tape WFSM:** weighted finite state machine jointly mapping between k strings

$\psi_1(x_1, x_2) = 13.26$

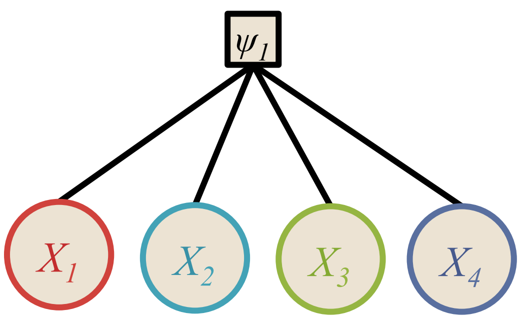
A **WFST** is a function that maps a pair of strings to a score.



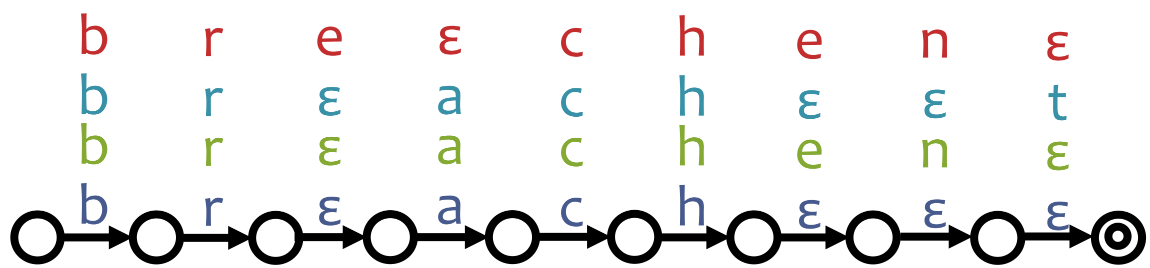
(Dreyer, Smith, & Eisner, 2008)

k-tape WFSM as a Factor Graph

- **WFSA:** weighted finite state **automata**
- **WFST:** weighted finite state **transducer**
- ***k*-tape WFSM:** weighted finite state machine jointly mapping between k strings



$\psi_1 =$



$\psi_1(x_1, x_2, x_3, x_4) = 13.26$

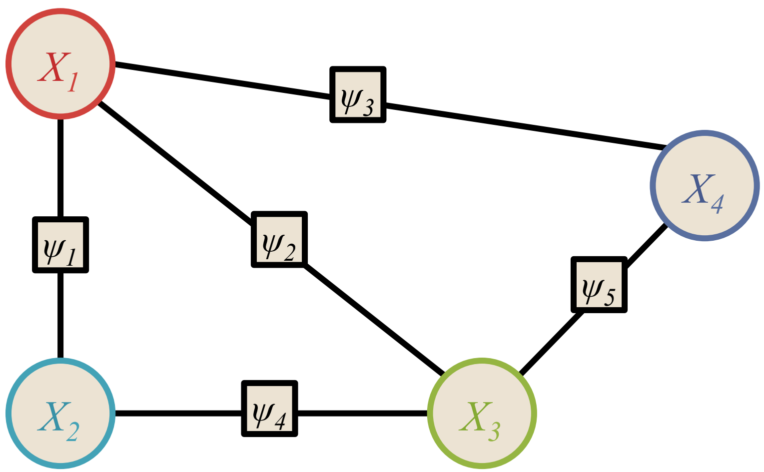
A ***k*-tape WFSM** is a function that maps ***k*** strings to a score.

What's wrong with a **100-tape WFSM** for jointly modeling the 100 distinct forms of a Polish verb?

- Each arc represents a 100-way edit operation
- Too many arcs!

Factor Graphs over Multiple Strings

$$P(x_1, x_2, x_3, x_4) = 1/Z\ \psi_1(x_1, x_2)\ \psi_2(x_1, x_3)\ \psi_3(x_1, x_4)\ \psi_4(x_2, x_3)\ \psi_5(x_3, x_4)$$

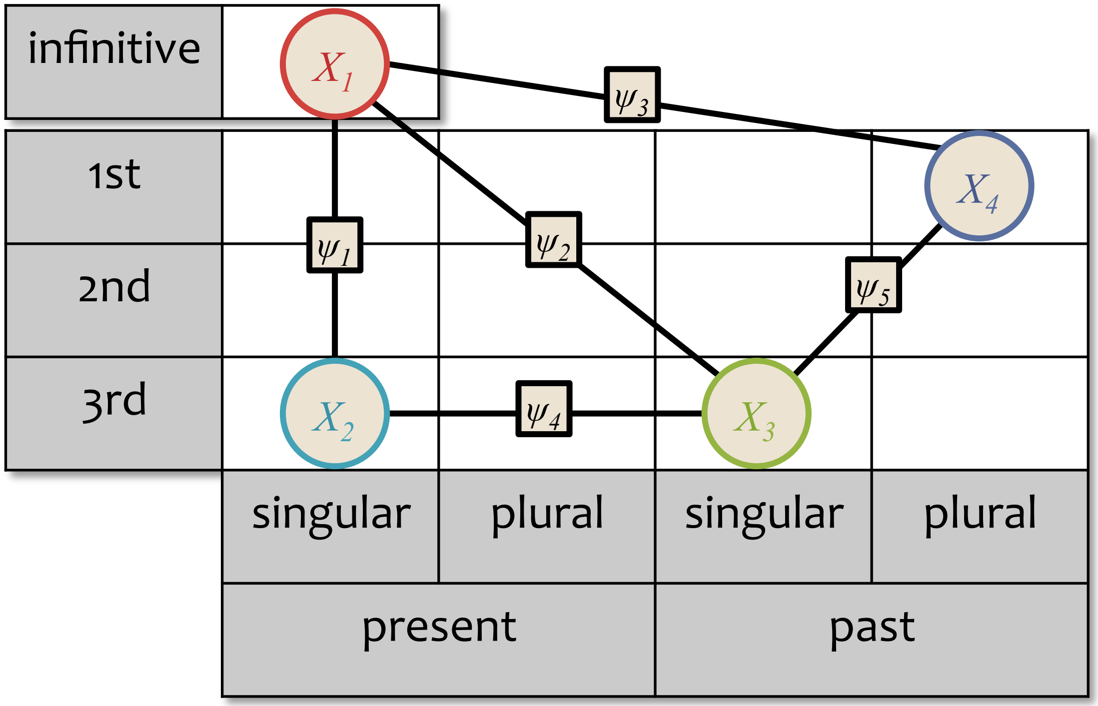


Instead, just build factor graphs with WFST factors (i.e. factors of arity 2)

(Dreyer & Eisner, 2009)

Factor Graphs over Multiple Strings

$$P(x_1, x_2, x_3, x_4) = 1/Z\ \psi_1(x_1, x_2)\ \psi_2(x_1, x_3)\ \psi_3(x_1, x_4)\ \psi_4(x_2, x_3)\ \psi_5(x_3, x_4)$$

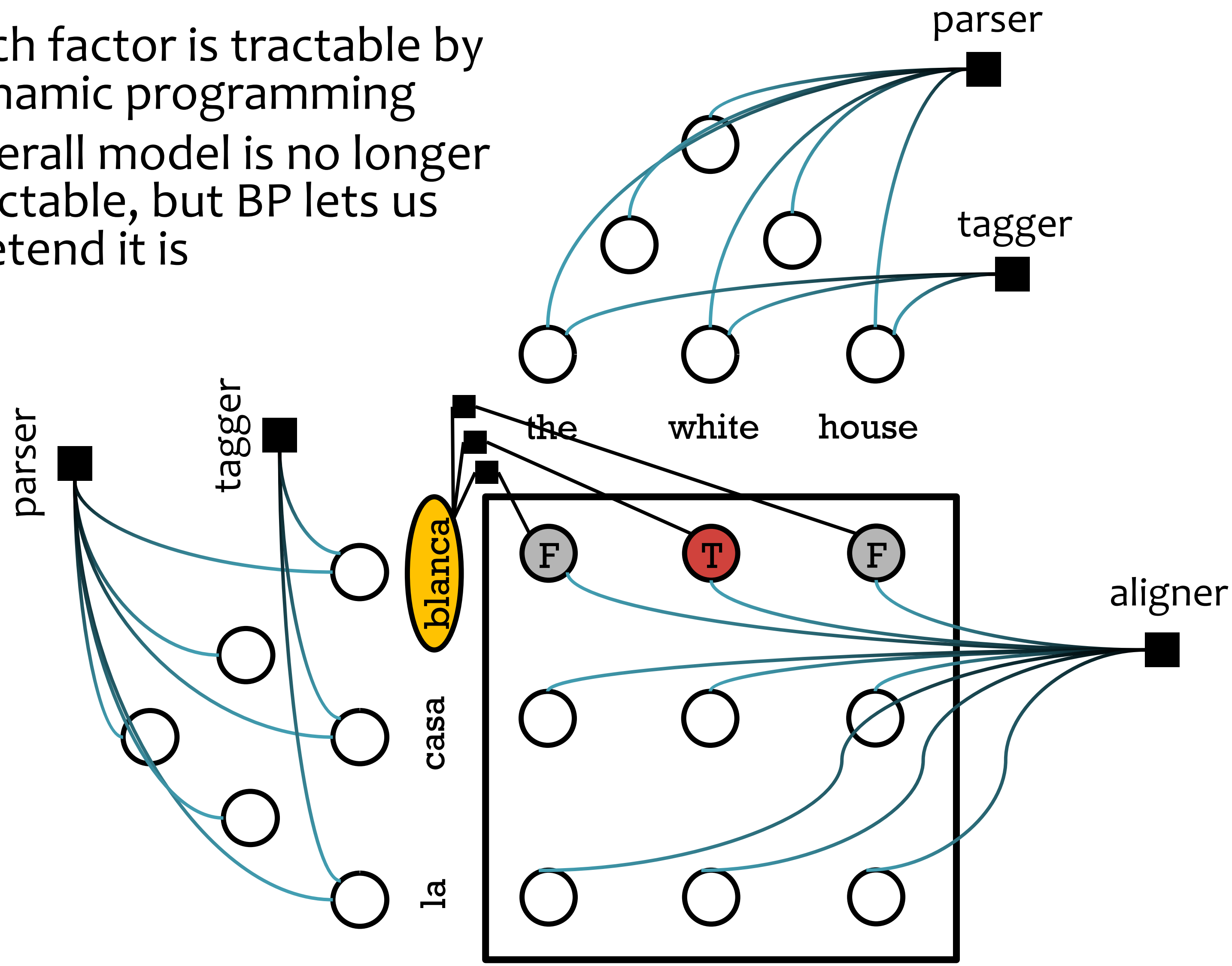


Instead, just build factor graphs with WFST factors (i.e. factors of arity 2)

(Dreyer & Eisner, 2009)

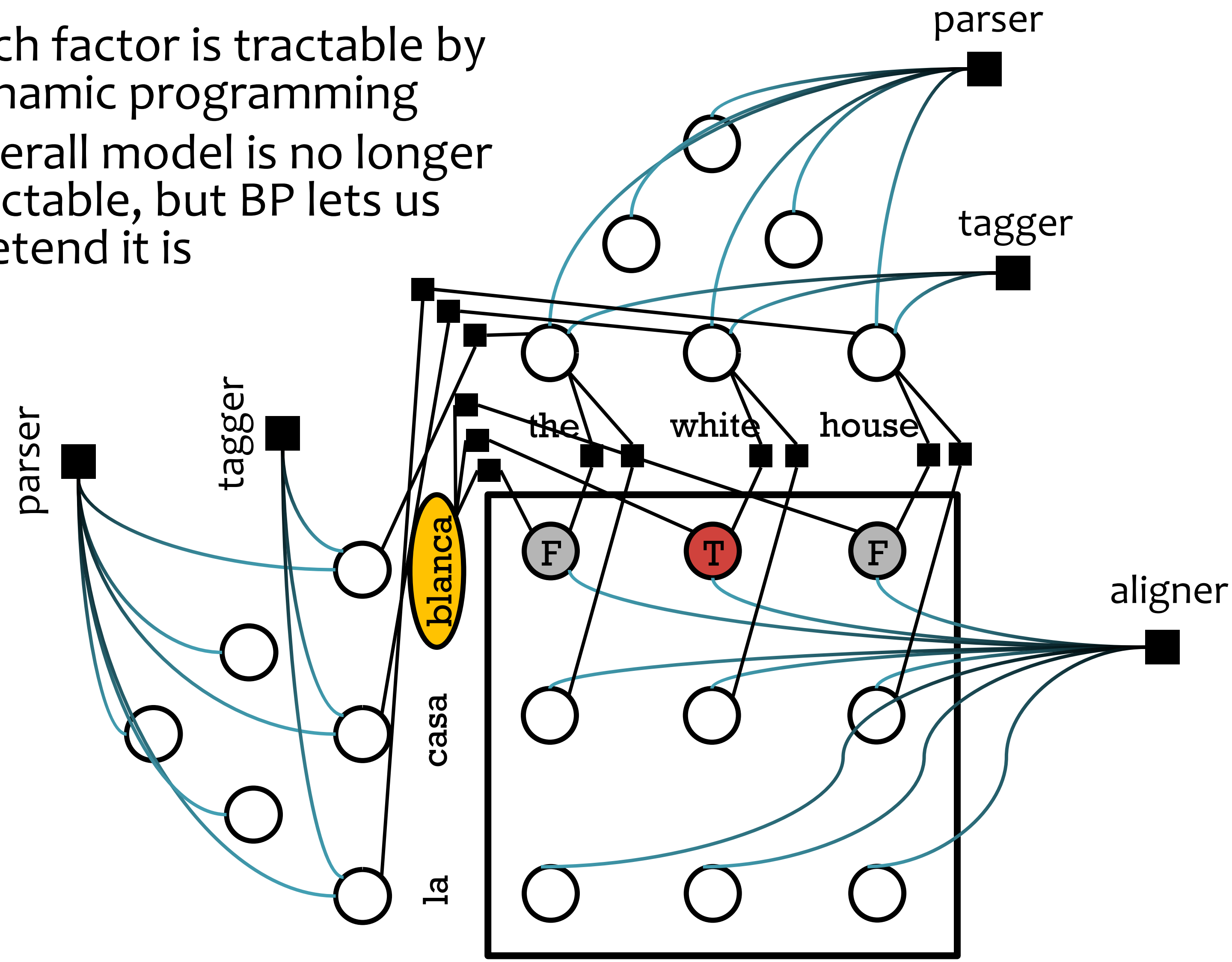
BP for Coordination of Algorithms

- Each factor is tractable by dynamic programming
- Overall model is no longer tractable, but BP lets us pretend it is



BP for Coordination of Algorithms

- Each factor is tractable by dynamic programming
- Overall model is no longer tractable, but BP lets us pretend it is



Section 5:
What if even BP is slow?

Computing fewer message updates

Computing them faster

Outline

- Do you want to push past the simple NLP models (logistic regression, PCFG, etc.) that we've all been using for 20 years?
- Then this tutorial is extremely practical for you!
  1. **Models:** Factor graphs can express interactions among linguistic structures.
  2. **Algorithm:** BP estimates the global effect of these interactions on each variable, using local computations.
  3. **Intuitions:** What's going on here? Can we trust BP's estimates?
  4. **Fancier Models:** Hide a whole grammar and dynamic programming algorithm within a single factor. BP coordinates multiple factors.
  5. **Tweaked Algorithm:** Finish in fewer steps and make the steps faster.
  6. **Learning:** Tune the parameters. Approximately improve the true predictions -- or truly improve the approximate predictions.
  7. **Software:** Build the model you want!

Outline

- Do you want to push past the simple NLP models (logistic regression, PCFG, etc.) that we've all been using for 20 years?
- Then this tutorial is extremely practical for you!
  1. **Models:** Factor graphs can express interactions among linguistic structures.
  2. **Algorithm:** BP estimates the global effect of these interactions on each variable, using local computations.
  3. **Intuitions:** What's going on here? Can we trust BP's estimates?
  4. **Fancier Models:** Hide a whole grammar and dynamic programming algorithm within a single factor. BP coordinates multiple factors.
  5. **Tweaked Algorithm:** Finish in fewer steps and make the steps faster.
  6. **Learning:** Tune the parameters. Approximately improve the true predictions -- or truly improve the approximate predictions.
  7. **Software:** Build the model you want!

Loopy Belief Propagation Algorithm

1. For every directed edge, initialize its message to the uniform distribution.
2. Repeat until all normalized beliefs converge:
   a. **Pick** a directed edge u → v.
   b. **Update** its message: recompute u → v from its "parent" messages v' → u for v' ≠ v.

Or if u has high degree, can share work for speed:

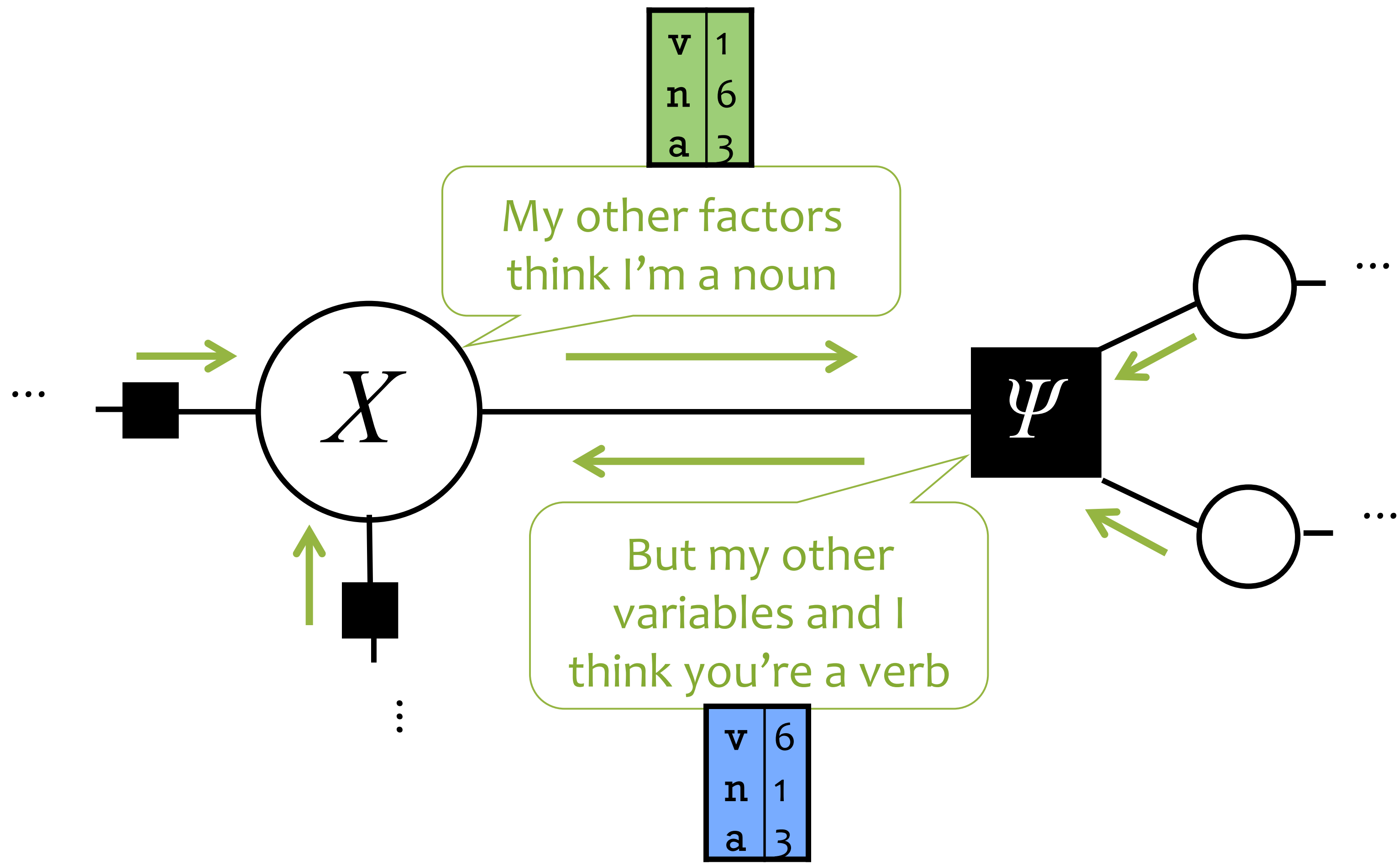
- Compute *all* outgoing messages u → … at once, based on all incoming messages … → u.

Loopy Belief Propagation Algorithm

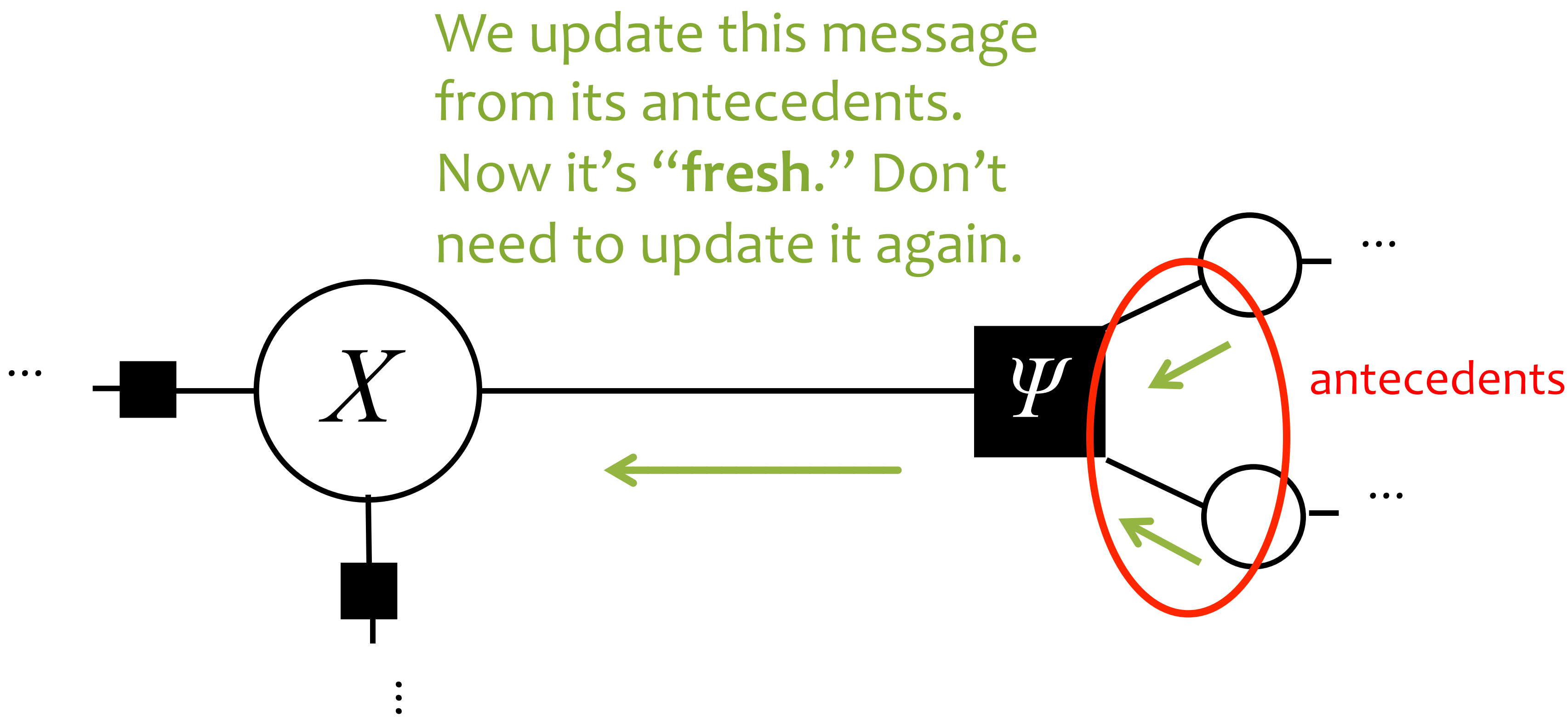
1. For every directed edge, initialize its message to the uniform distribution.
2. Repeat until all normalized beliefs converge:
   a. **Pick** a directed edge u → v.
   b. **Update** its message: recompute u → v from its "parent" messages v' → u for v' ≠ v.

Which edge do we pick and recompute?
A "stale" edge?

Message Passing in Belief Propagation

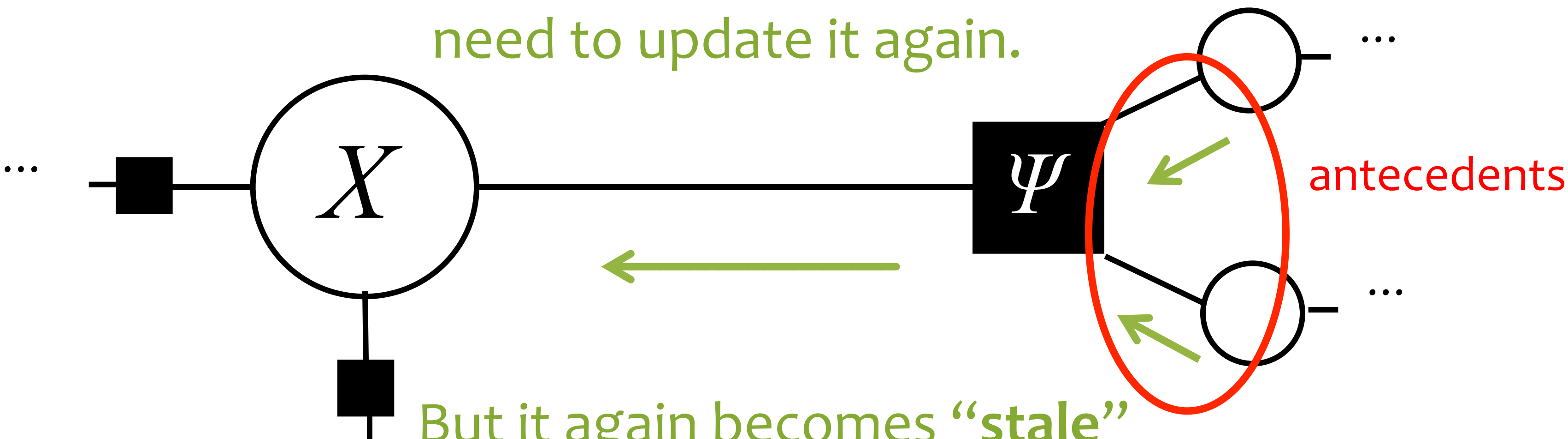


Stale Messages



Stale Messages

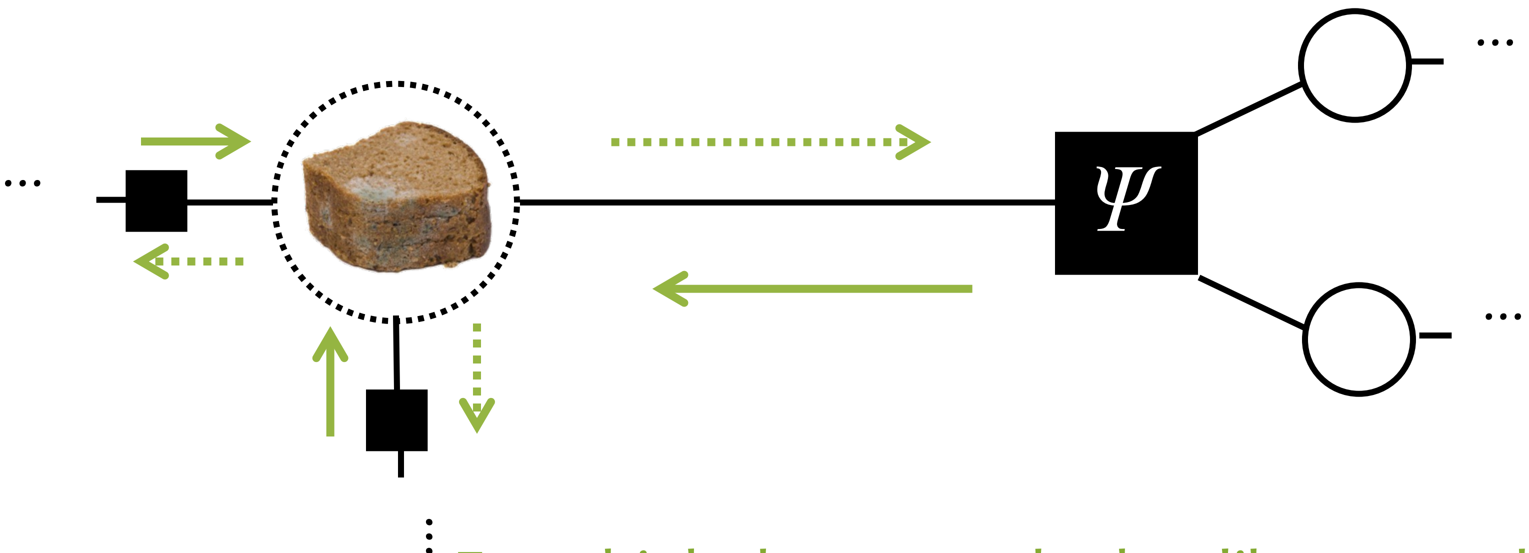
We update this message from its antecedents. Now it's "**fresh.**" Don't need to update it again.



But it again becomes "**stale**" – out of sync with its antecedents – if <u>they</u> change. Then we do need to revisit.

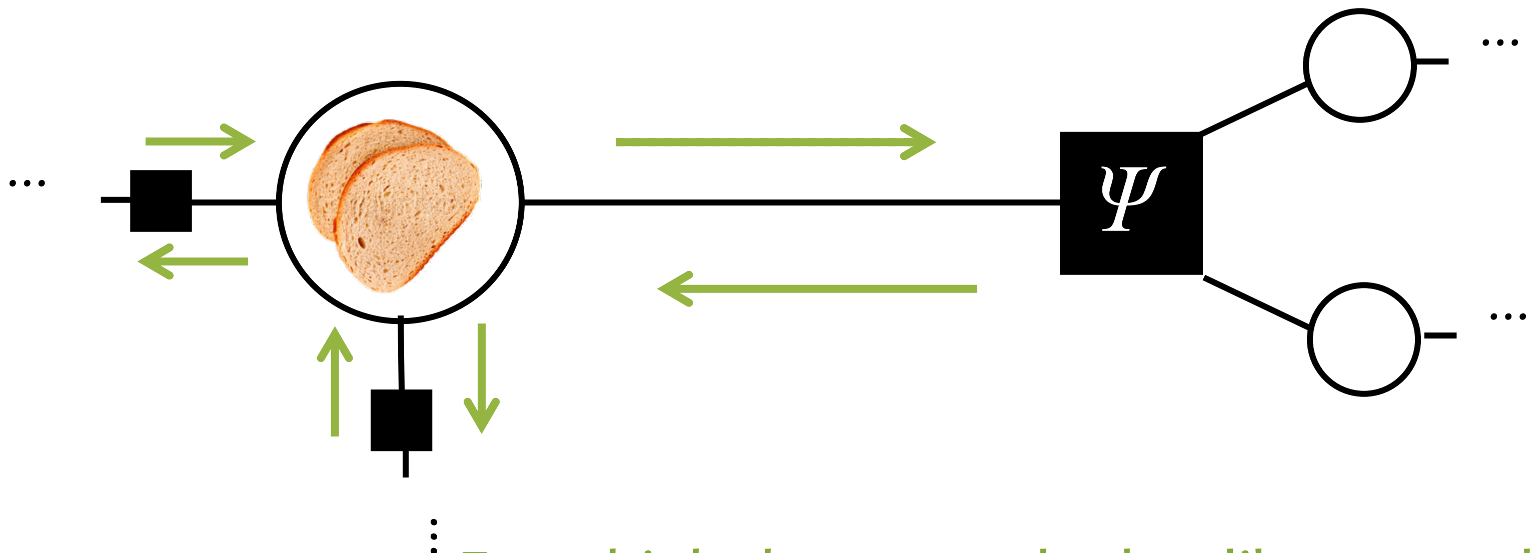
The edge is ***very stale*** *if its antecedents have changed* ***a lot*** *since its last update. Especially in a way that might make* ***this*** *edge change a lot.*

Stale Messages



For a high-degree node that likes to update all its outgoing messages at once …
We say that the whole node is very stale if its incoming messages have changed a lot.

Stale Messages

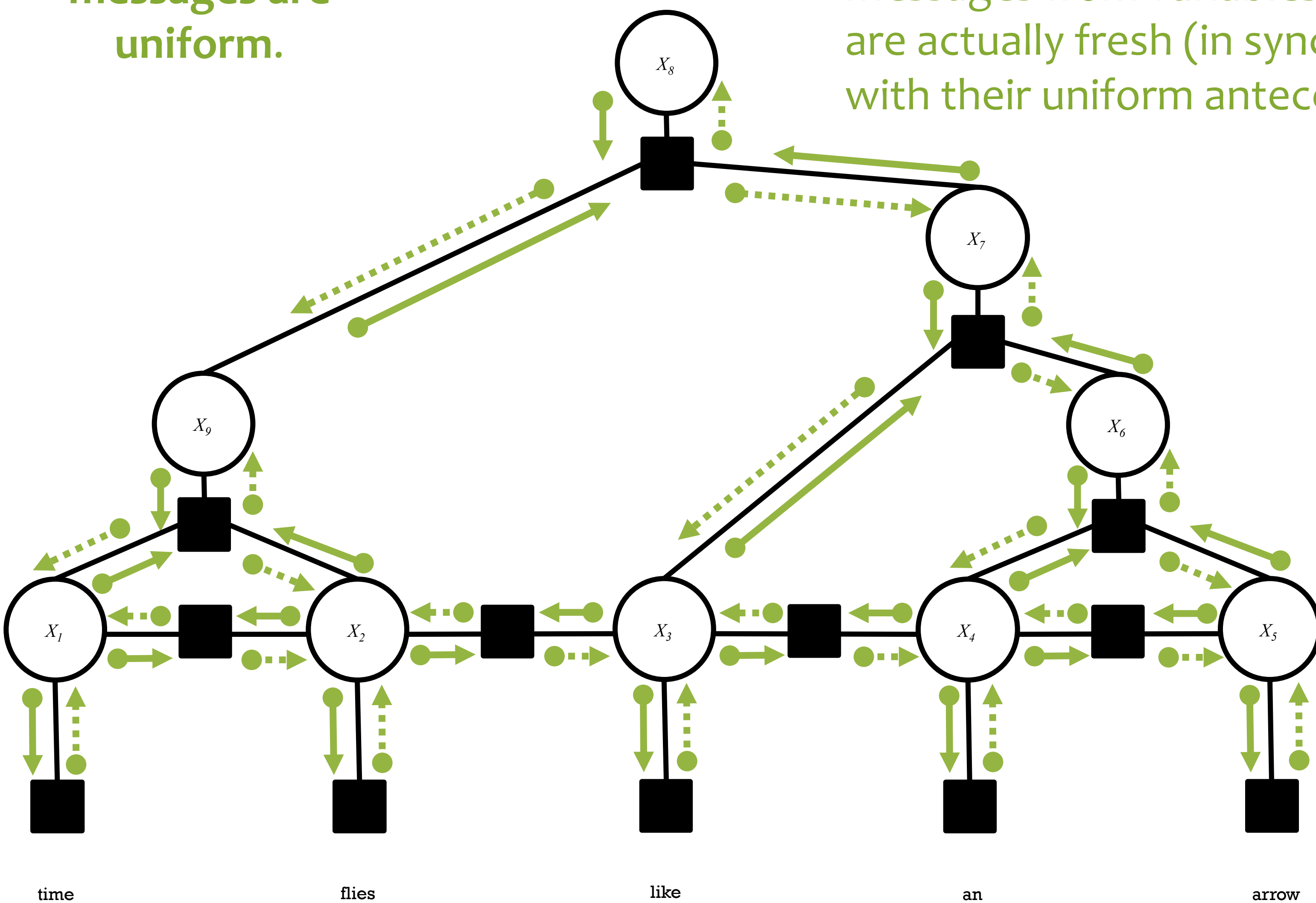


For a high-degree node that likes to update all its outgoing messages at once …
We say that the whole node is very stale if its incoming messages have changed a lot.

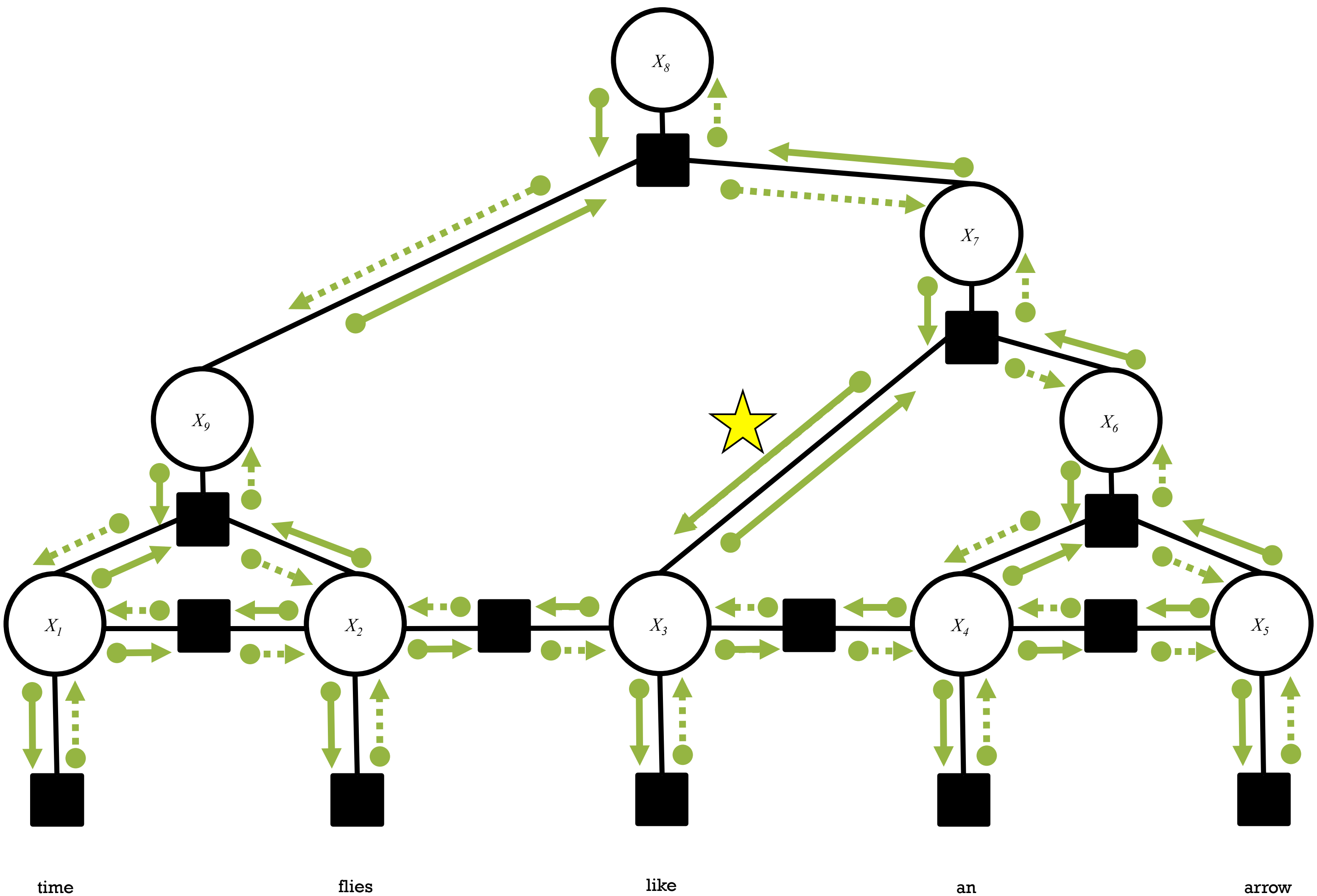
Maintain an Queue of Stale Messages to Update

Initially all messages are uniform.

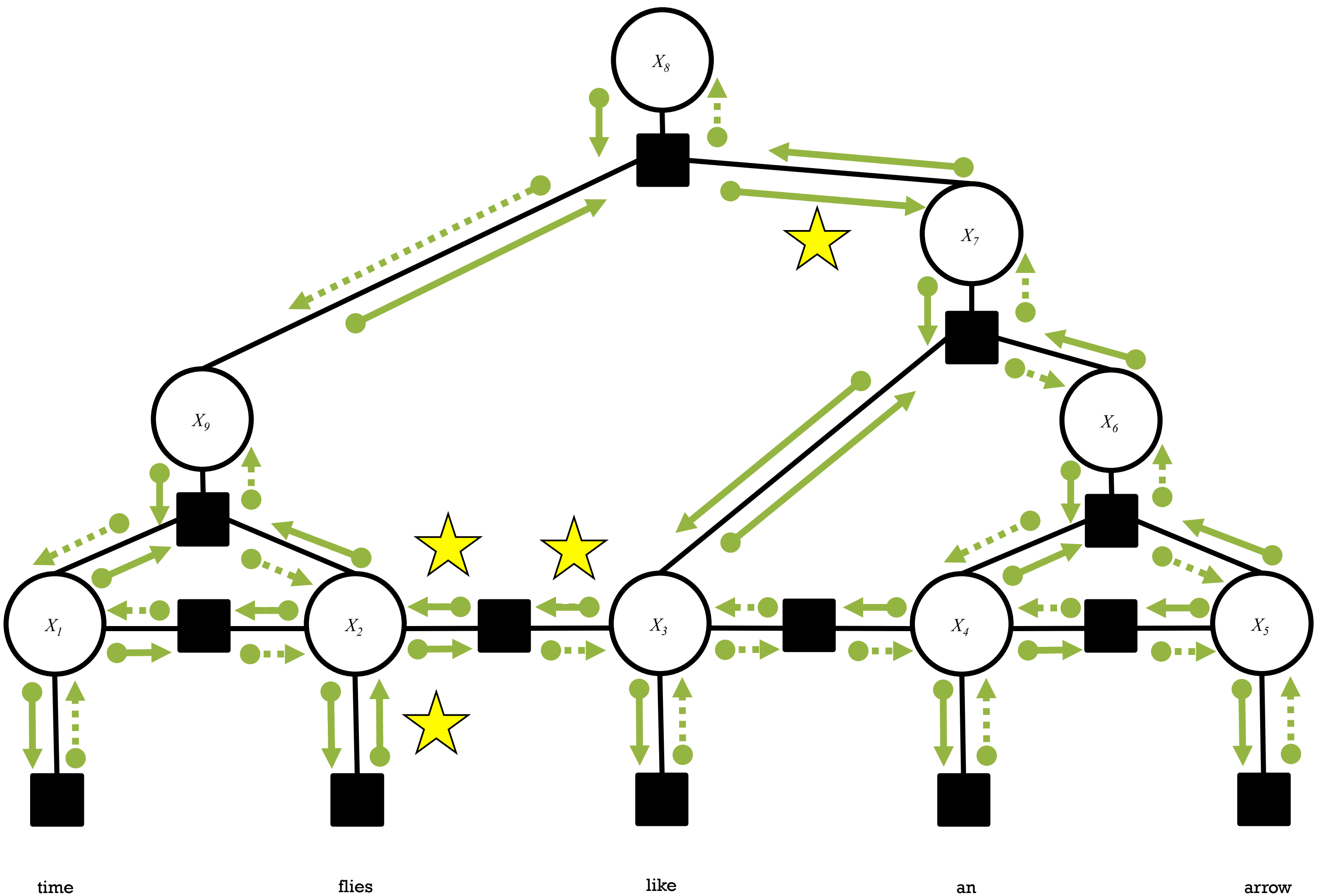
Messages from factors are stale.
Messages from variables are actually fresh (in sync with their uniform antecedents).



Maintain an Queue of Stale Messages to Update



Maintain an Queue of Stale Messages to Update



Maintain ~~an Queue~~ of Stale Messages to Update

a priority queue! (heap)

- Residual BP: **Always update the message that is *most stale*** (would be *most changed* by an update).
- Maintain a priority queue of stale edges (& perhaps variables).
  - Each step of residual BP: "Pop and update."
  - Prioritize by degree of staleness.
  - When something becomes stale, put it on the queue.
  - If it becomes staler, move it earlier on the queue.
  - Need a measure of staleness.
- So, process **biggest updates first.**
- Dramatically improves speed of convergence.
  - And chance of converging at all. ☺

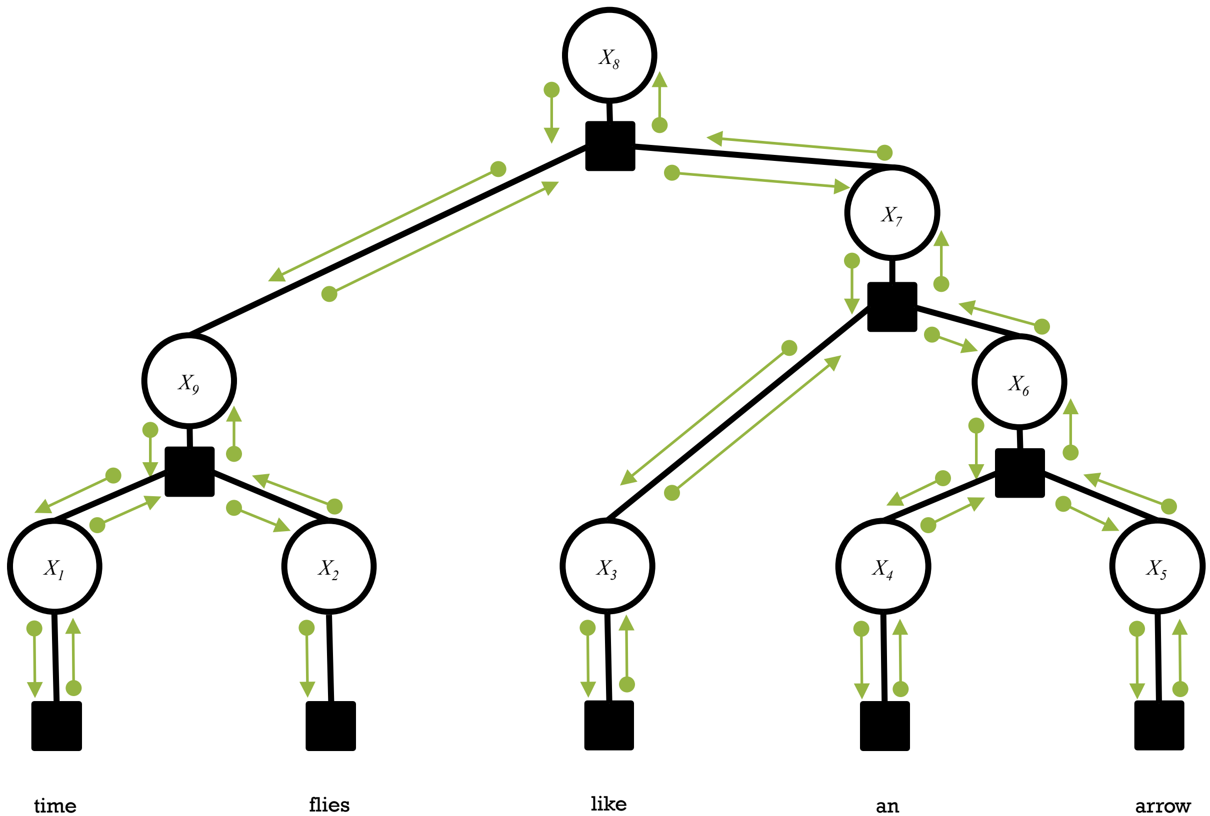
(Elidan et al., 2006)

But what about the topology?

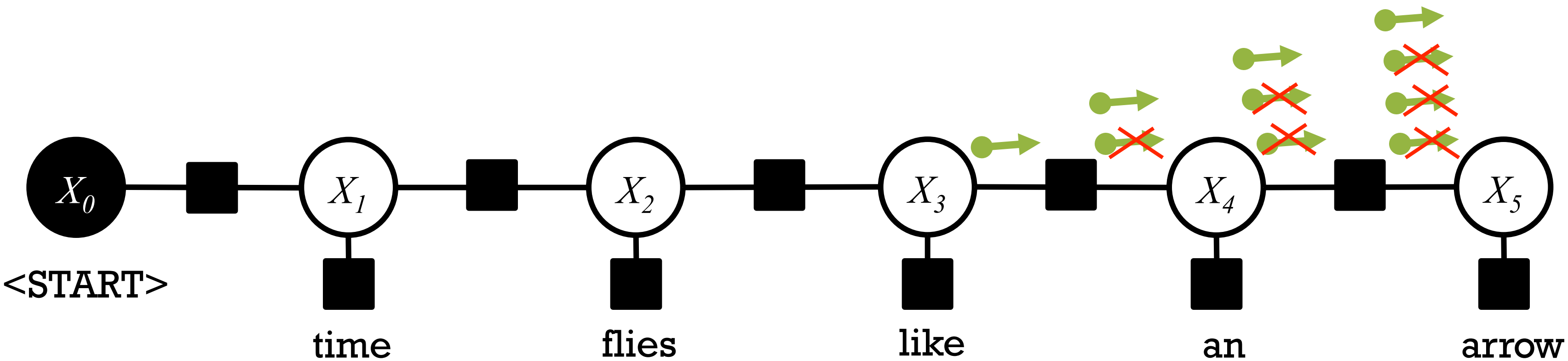
In a graph with no cycles:

1. Send messages from the **leaves** to the **root.**
2. Send messages from the **root** to the **leaves.**

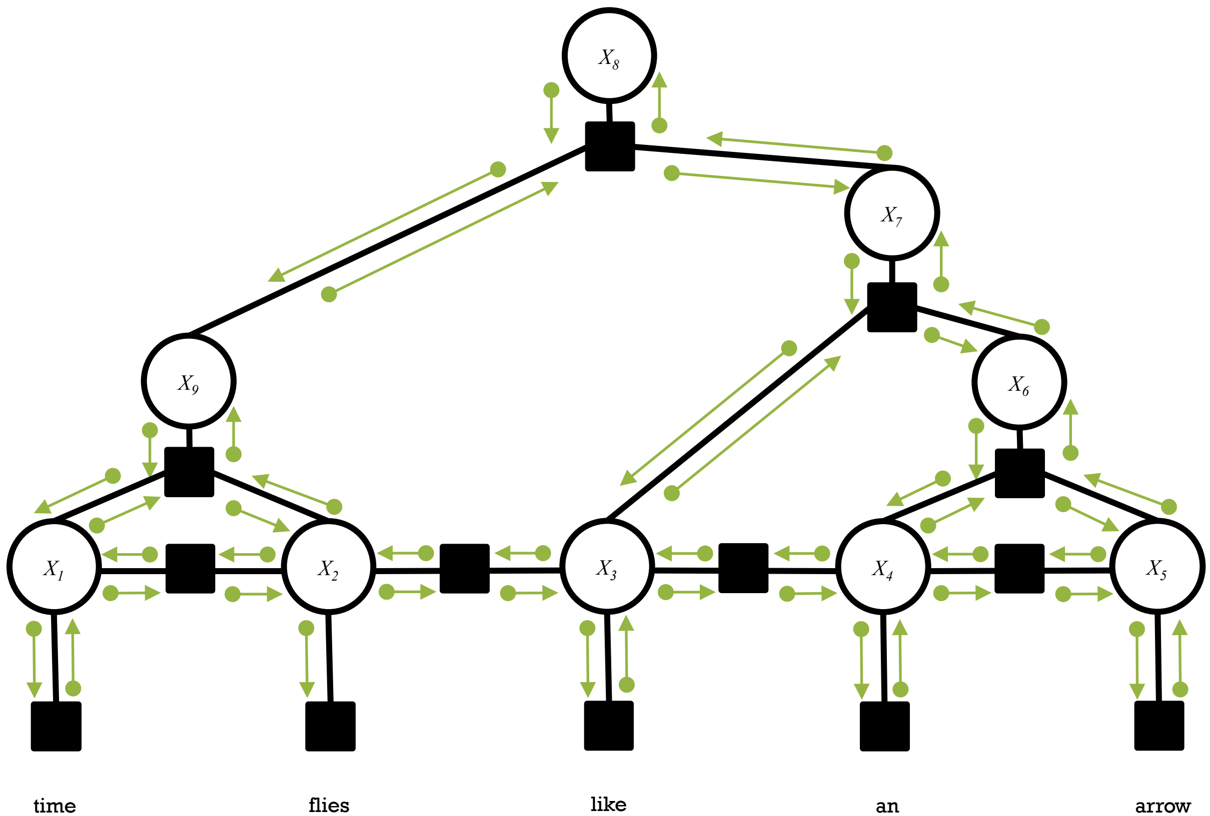
Each outgoing message is sent only after all its incoming messages have been received.



A bad update order for residual BP!



Try updating an entire acyclic subgraph

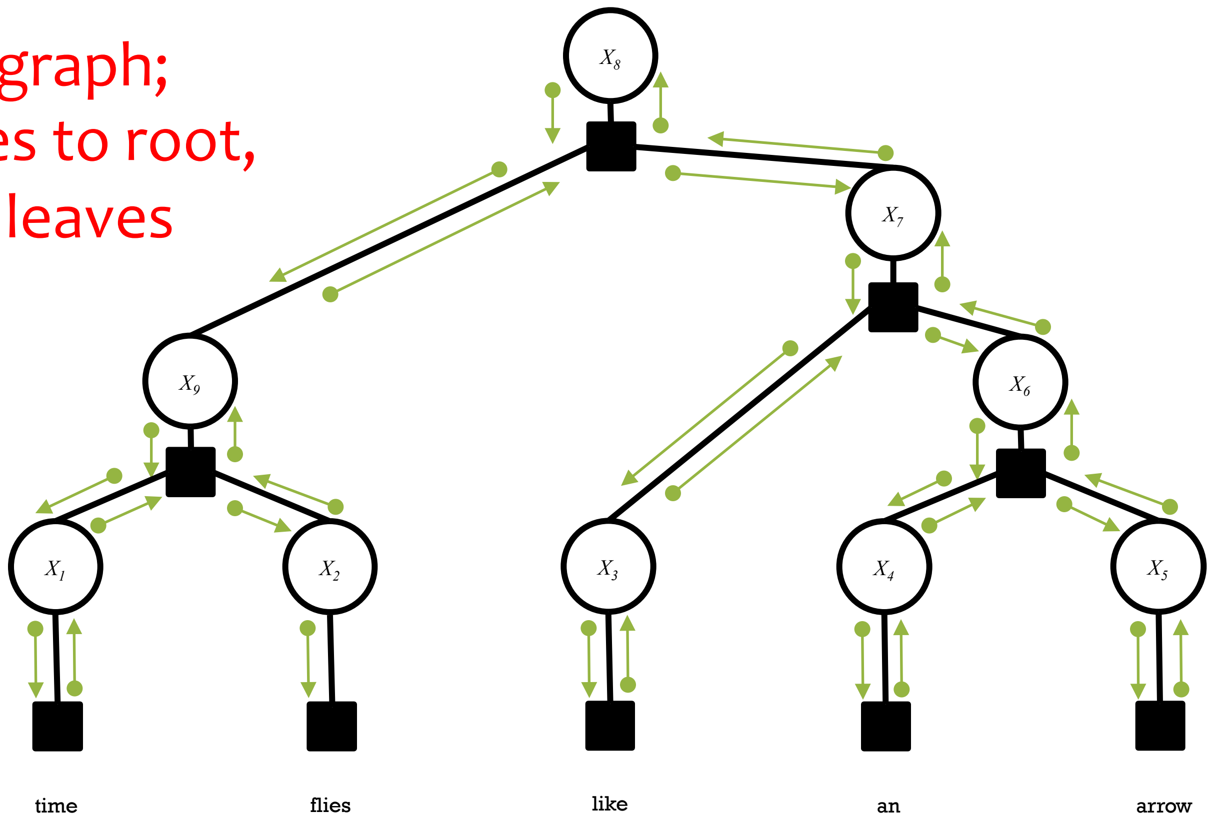


Tree-based Reparameterization (Wainwright et al. 2001); also see Residual Splash

Johns Hopkins University

Try updating an entire acyclic subgraph

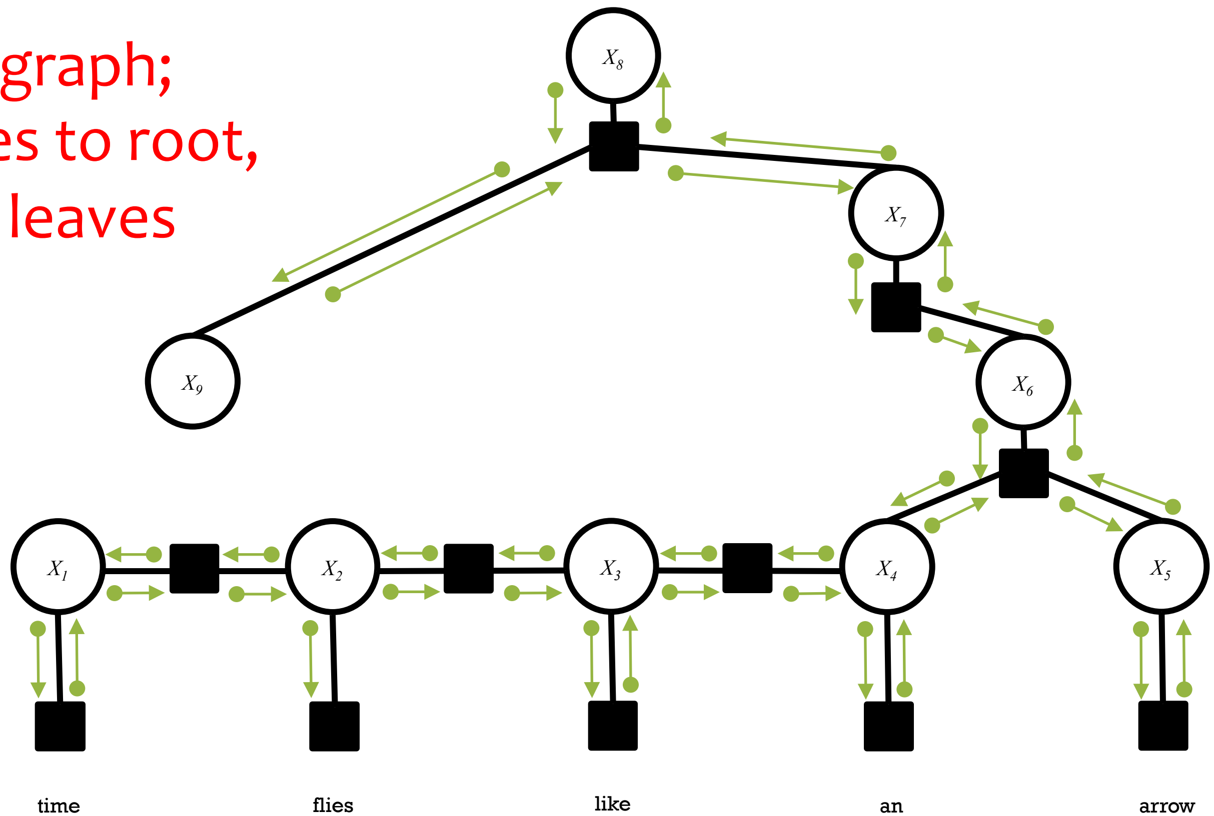
Pick this subgraph;
update leaves to root,
then root to leaves



Tree-based Reparameterization (Wainwright et al. 2001); also see Residual Splash

Try updating an entire acyclic subgraph

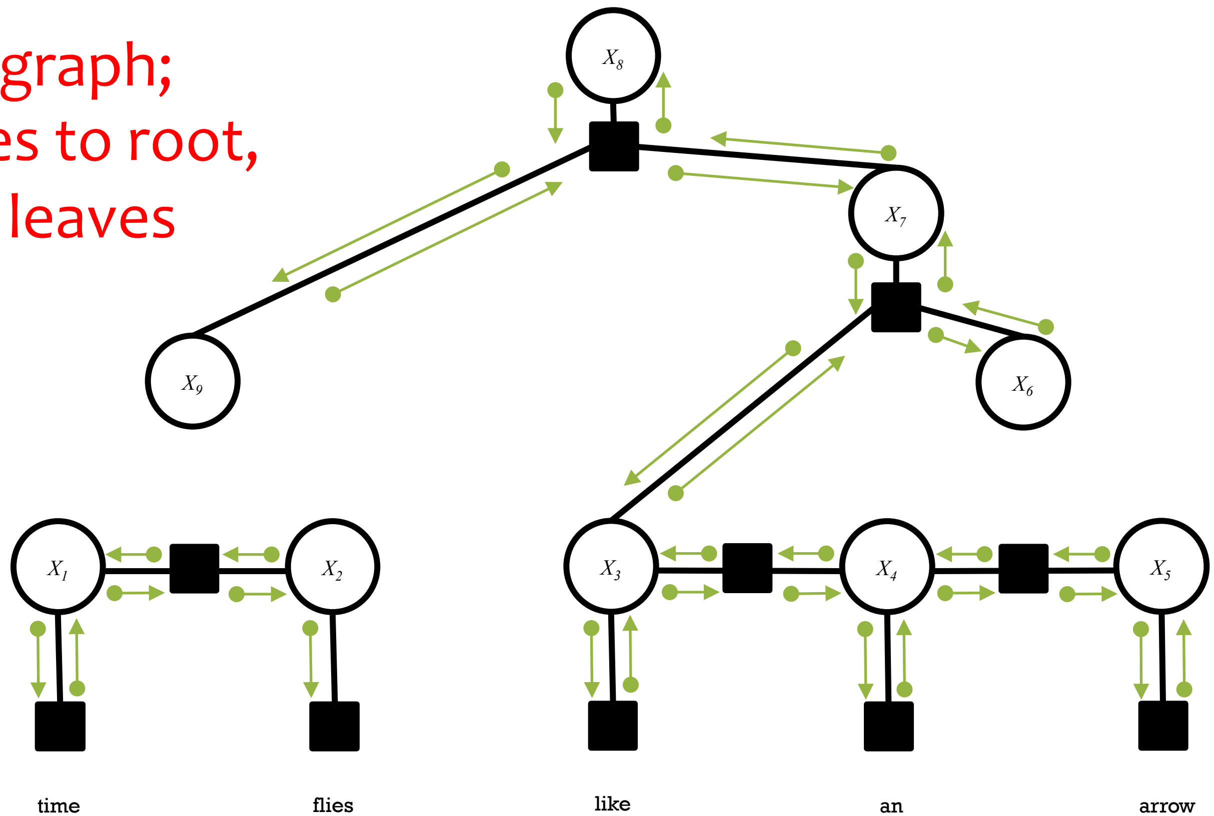
Another subgraph;
update leaves to root,
then root to leaves



Tree-based Reparameterization (Wainwright et al. 2001); also see Residual Splash

Try updating an entire acyclic subgraph

Another subgraph; update leaves to root, then root to leaves



At every step, pick a spanning tree (or spanning forest) that covers **many stale edges**

As we update messages in the tree, it affects staleness of messages outside the tree

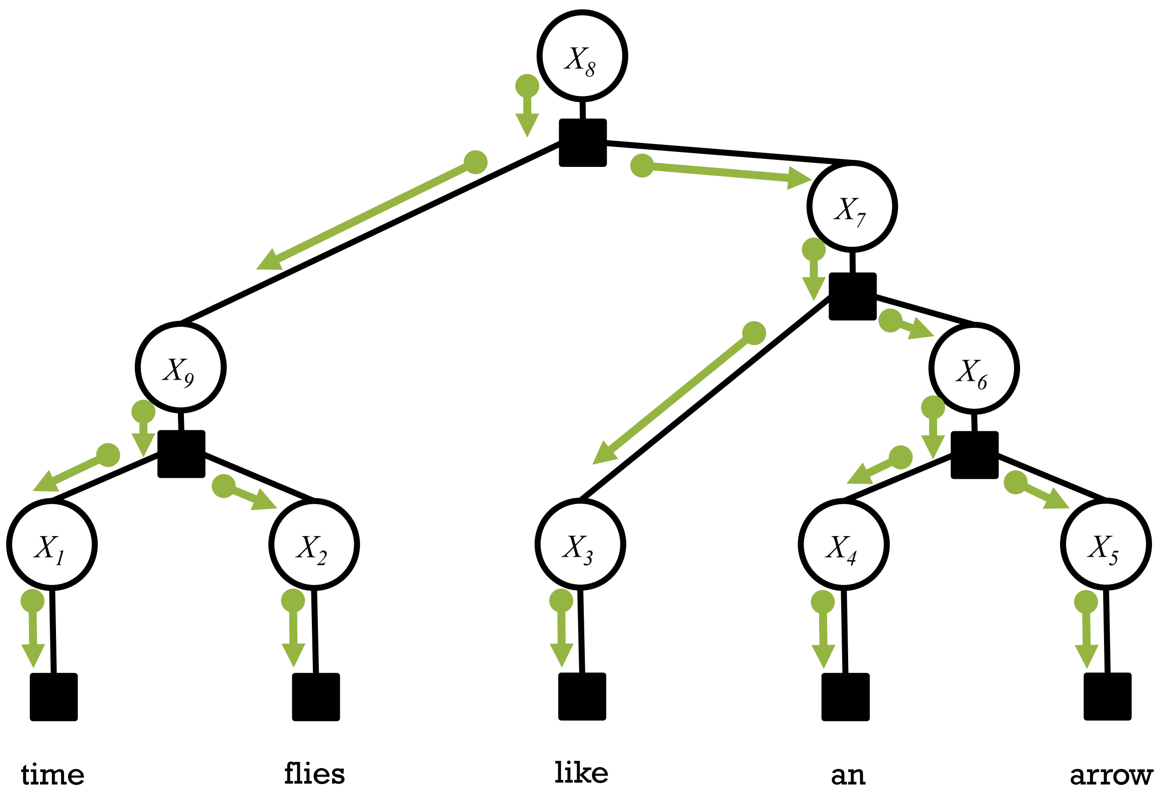
Tree-based Reparameterization (Wainwright et al. 2001); also see Residual Splash

Acyclic Belief Propagation

In a graph with no cycles:

1. Send messages from the **leaves** to the **root.**
2. Send messages from the **root** to the **leaves.**

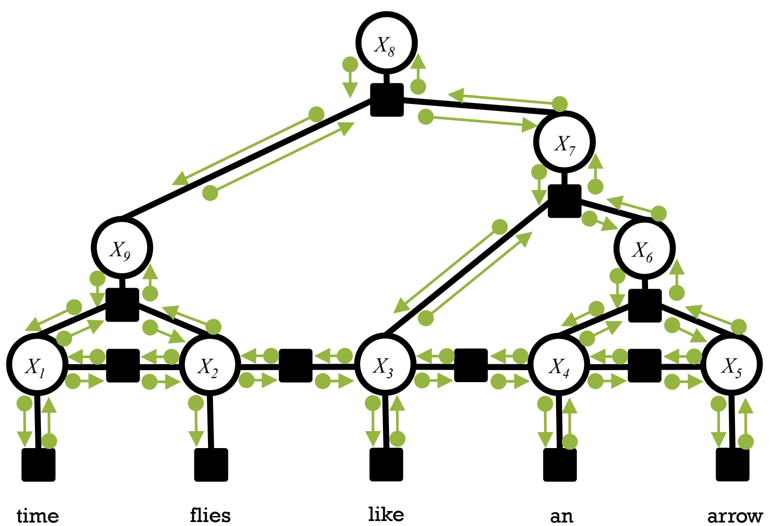
Each outgoing message is sent only after all its incoming messages have been received.



Summary of Update Ordering

In what order do we send messages for Loopy BP?

- Asynchronous
  - Pick a directed edge: update its message
  - Or, pick a vertex: update *all* its outgoing messages at once



Wait for your antecedents

Don't update a message if its antecedents will get a big update.

Otherwise, will have to re-update.

- **Size.** Send big updates first.
  - Forces other messages to wait for them.
- **Topology.** Use graph structure.
  - E.g., in an acyclic graph, a message can wait for *all* updates before sending.

Message Scheduling

The order in which messages are sent has a significant effect on convergence

- Synchronous (bad idea)
  - Compute all the messages
  - Send all the messages
- Asynchronous
  - Pick an edge: compute and send that message
- Tree-based Reparameterization
  - Successively update embedded spanning trees
  - Choose spanning trees such that each edge is included in at least one
- Residual BP
  - Pick the edge whose message would change the most if sent: compute and send that message

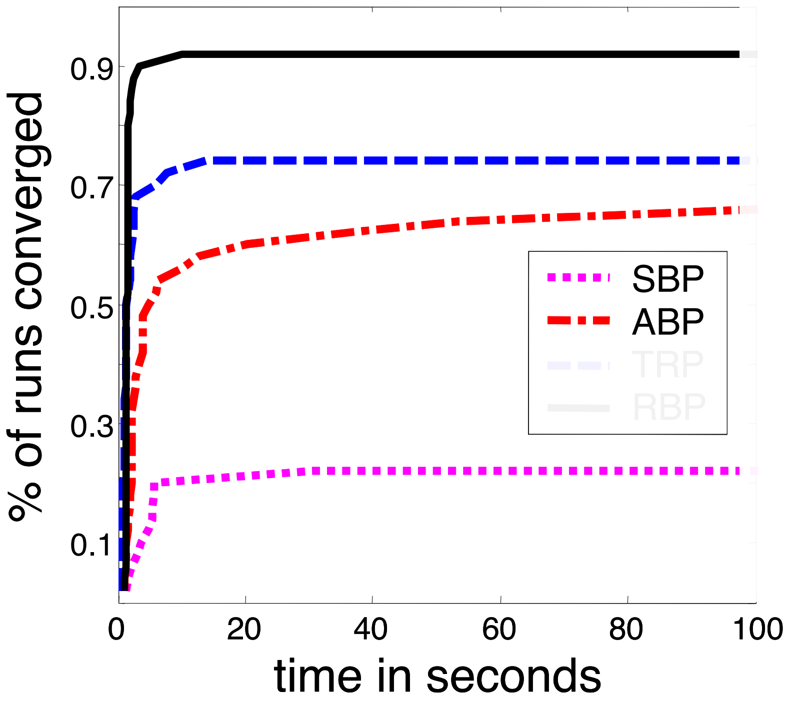
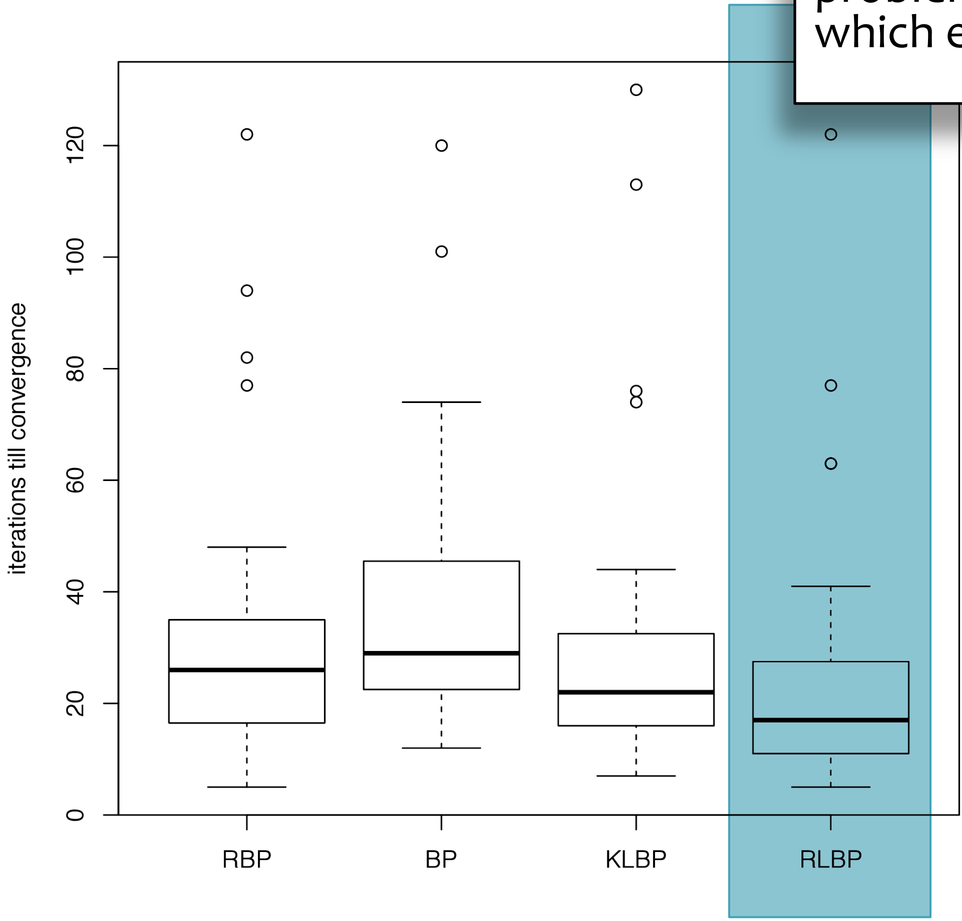


Figure from (Elidan, McGraw, & Koller, 2006)

Message Scheduling



Even better dynamic scheduling may be possible by **reinforcement learning** of a problem-specific heuristic for choosing which edge to update next.

(Jiang, Moon, Daumé III, & Eisner, 2013)

Section 5:
What if even BP is slow?

Computing fewer message updates

$\boxed{\text{Computing them faster}}$

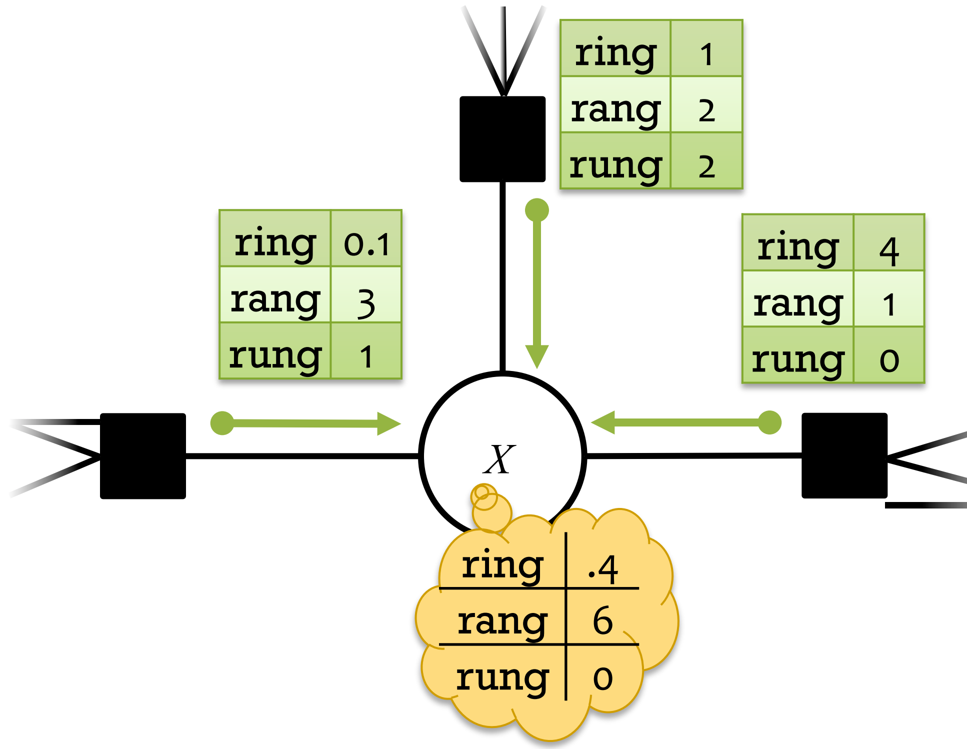
A variable has k possible values.
What if k is large or infinite?

Computing Variable Beliefs

Suppose…

- X_i is a discrete variable
- Each incoming messages is a Multinomial

Pointwise product is easy when the variable's domain is small and discrete



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

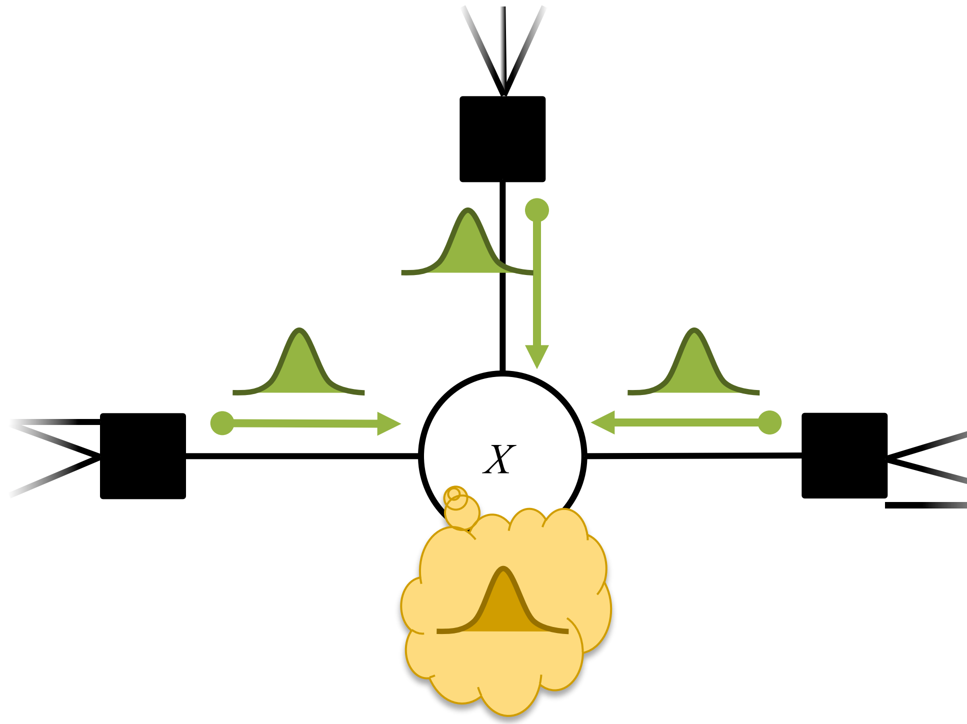
Computing Variable Beliefs

Suppose…

- X_i is a real-valued variable
- Each incoming message is a Gaussian

The pointwise product of n Gaussians is…

…a Gaussian!

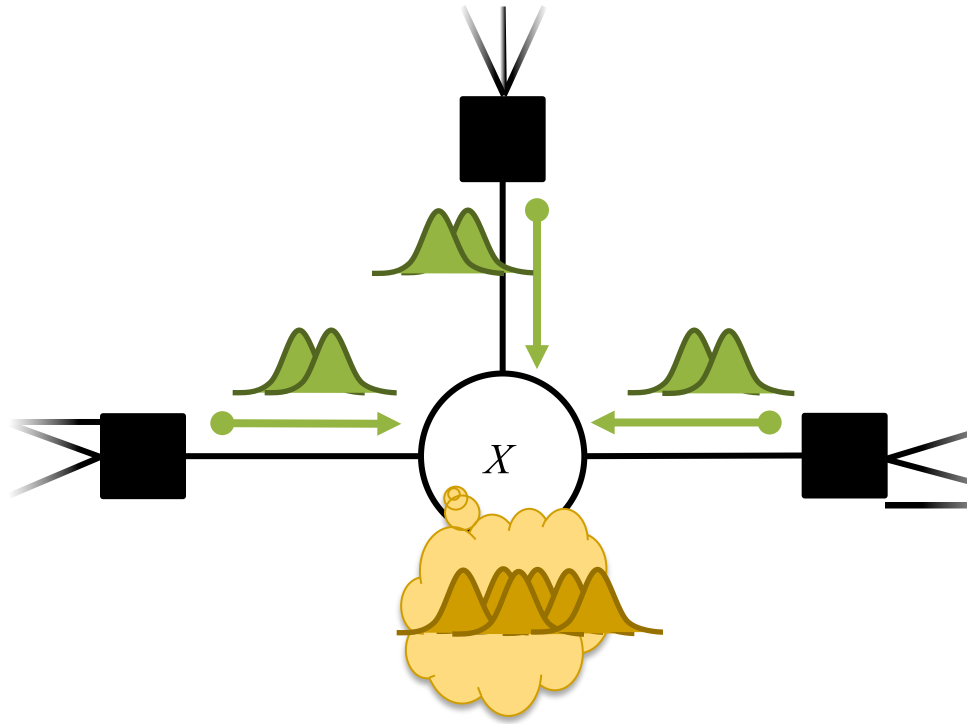


$$b_i(x_i) = \prod_{\alpha \in \mathcal{N}(i)} \mu_{\alpha \to i}(x_i)$$

Computing Variable Beliefs

Suppose…

- X_i is a real-valued variable
- Each incoming messages is a mixture of k Gaussians



The pointwise product explodes!

$$p(x) = \underbrace{p_1(x)}_{\substack{(\ 0.3\, q_{1,1}(x) \\ +\ 0.7\, q_{1,2}(x))}} \underbrace{p_2(x)}_{\substack{(\ 0.5\, q_{2,1}(x) \\ +\ 0.5\, q_{2,2}(x))}} \ldots p_n(x)$$

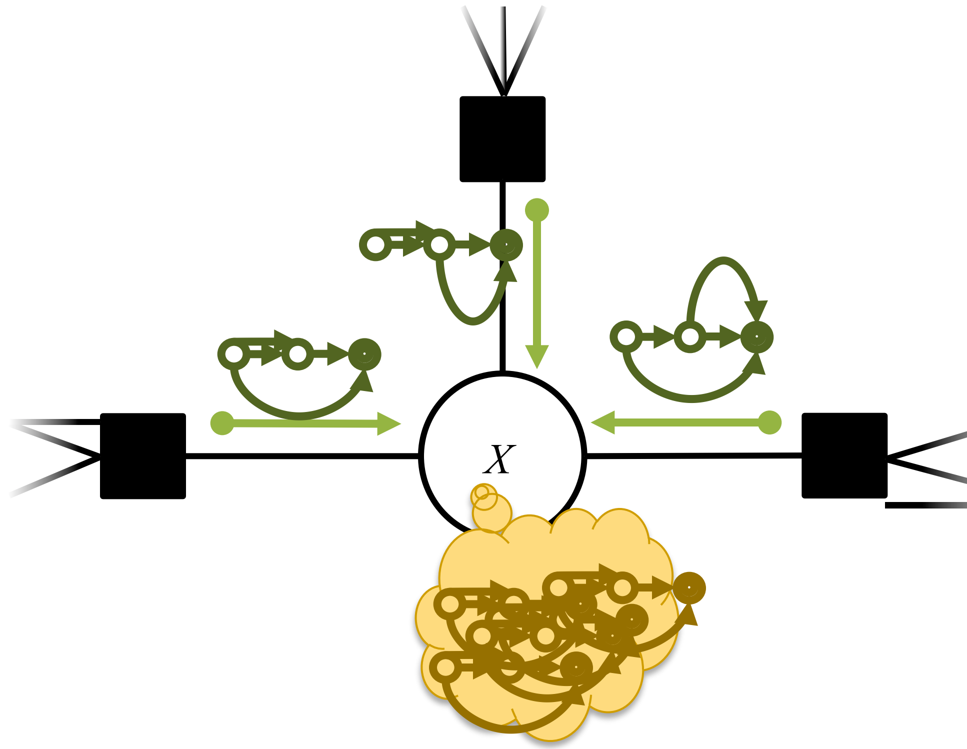
$$\boxed{b_i(x_i) = \prod_{\alpha \in \mathcal{N}(i)} \mu_{\alpha \to i}(x_i)}$$

Computing Variable Beliefs

Suppose…

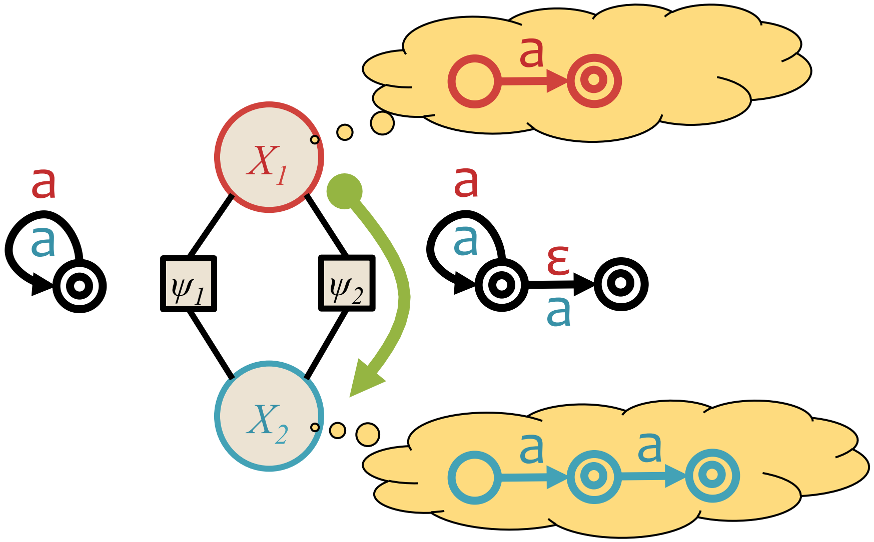
- X_i is a string-valued variable (i.e. its domain is the set of all strings)
- Each incoming messages is a FSA

The pointwise product explodes!



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

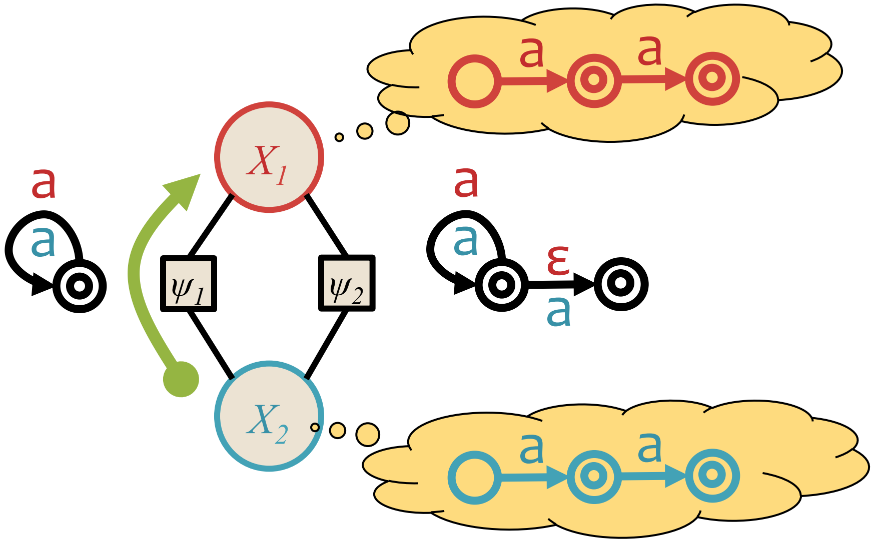
Example: String-valued Variables



- Messages can **grow larger** when sent through a transducer factor
- Repeatedly sending messages through a transducer can cause them to **grow to unbounded size**!

(Dreyer & Eisner, 2009)

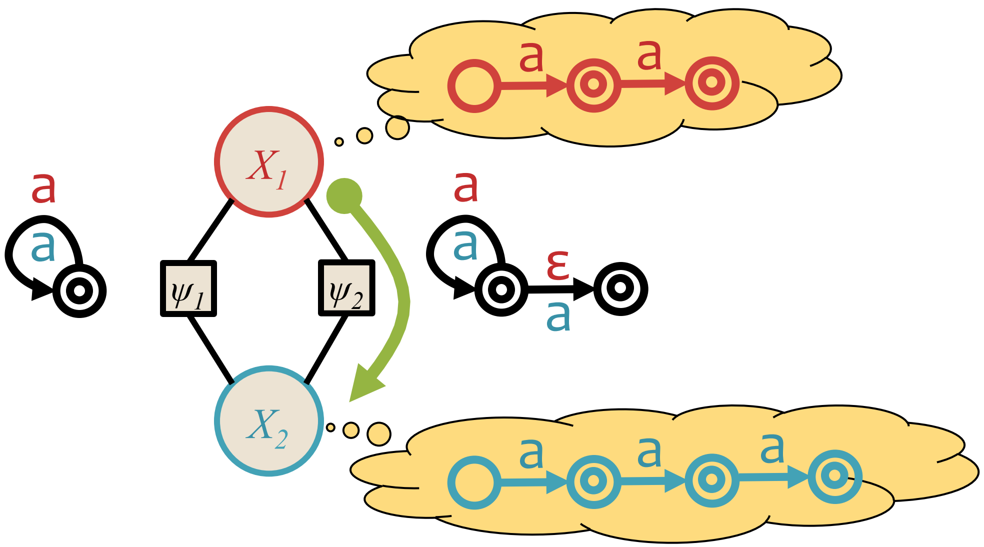
Example: String-valued Variables



- Messages can **grow larger** when sent through a transducer factor
- Repeatedly sending messages through a transducer can cause them to **grow to unbounded size**!

(Dreyer & Eisner, 2009)

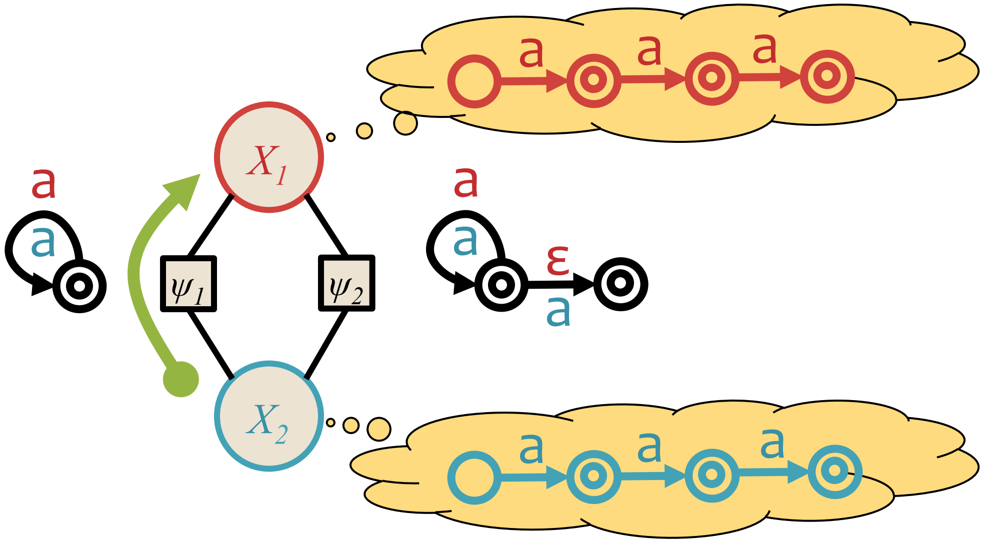
Example: String-valued Variables



- Messages can **grow larger** when sent through a transducer factor
- Repeatedly sending messages through a transducer can cause them to **grow to unbounded size**!

(Dreyer & Eisner, 2009)

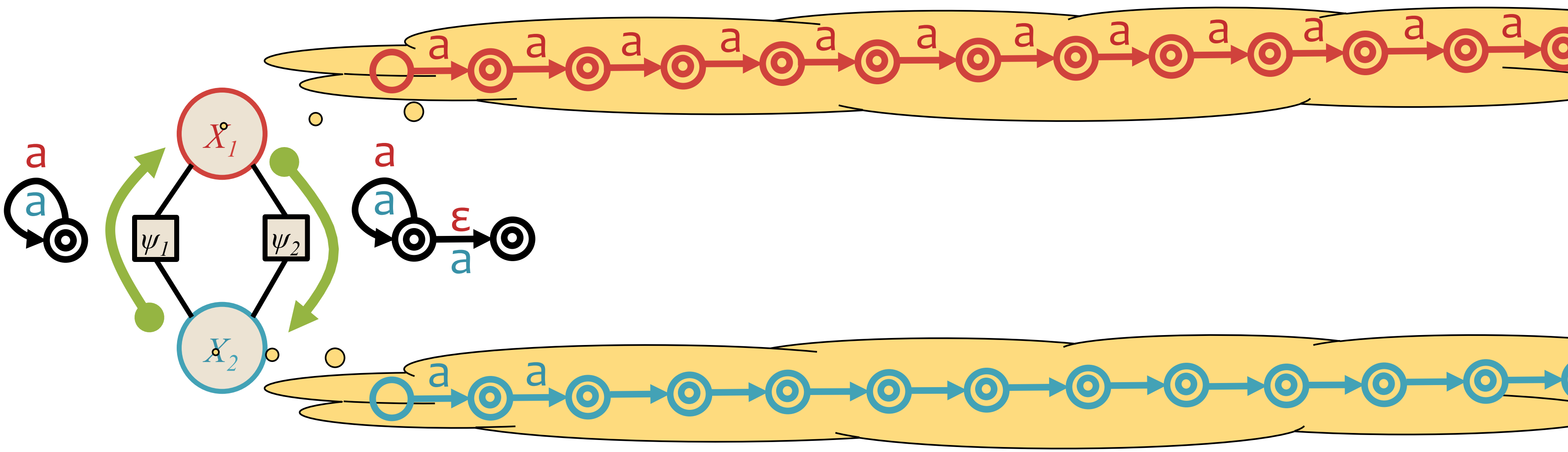
Example: String-valued Variables



- Messages can **grow larger** when sent through a transducer factor
- Repeatedly sending messages through a transducer can cause them to **grow to unbounded size**!

(Dreyer & Eisner, 2009)

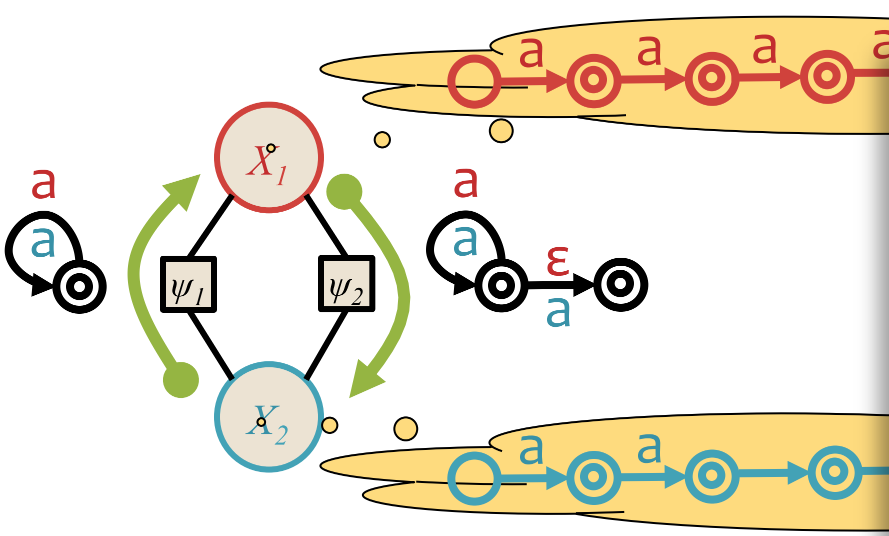
Example: String-valued Variables



- Messages can **grow larger** when sent through a transducer factor
- Repeatedly sending messages through a transducer can cause them to **grow to unbounded size**!

(Dreyer & Eisner, 2009)

Example: String-valued Variables



- The domain of these variables is **infinite** (i.e. Σ^*);
- WSFA's representation is **finite – but the size of the representation** can grow
- In cases where the domain of each variable is **small and finite** this is not an issue

- Messages can **grow larger** when sent through a transducer factor
- Repeatedly sending messages through a transducer can cause them to **grow to unbounded size**!

(Dreyer & Eisner, 2009)

Message Approximations

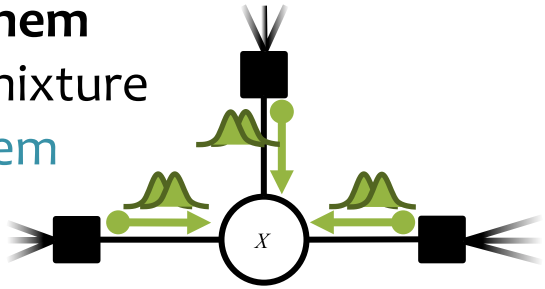
Three approaches to dealing with **complex messages:**

1. Particle Belief Propagation (see Section 3)
2. **Message pruning**
3. **Expectation propagation**

Message Pruning

- **Problem:** Product of d messages = complex distribution.
  - **Solution:** Approximate with a simpler distribution.
  - For speed, compute approximation *without* computing full product.

For **real variables,** try a mixture of K Gaussians:
- E.g., true product is a mixture of K^d Gaussians
  - **Prune back: Randomly keep just K of them**
  - Chosen in proportion to weight in full mixture
  - Gibbs sampling to efficiently choose them



- What if incoming messages are *not* Gaussian mixtures?
  - Could be anything sent by the factors …
  - Can extend technique to this case.

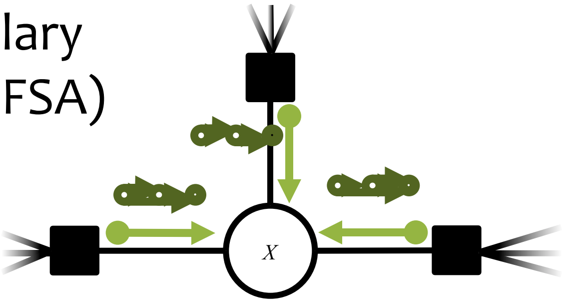
(Sudderth et al., 2002 –"Nonparametric BP")

Message Pruning

- **Problem:** Product of d messages = complex distribution.
  - **Solution:** Approximate with a simpler distribution.
  - For speed, compute approximation *without* computing full product.

For **string** variables, use a small finite set:

- Each message μ_i gives positive probability to …
  - … every word in a 50,000 word vocabulary
  - … every string in Σ^* (using a weighted FSA)



- **Prune back** to a list L of a few "good" strings
  - Each message adds its own K best strings to L
  - For each $x \in L$, let $\mu(x) = \prod_i \mu_i(x)$ – each message scores x
  - For each $x \notin L$, let $\mu(x) = 0$

(Dreyer & Eisner, 2009)

Expectation Propagation (EP)

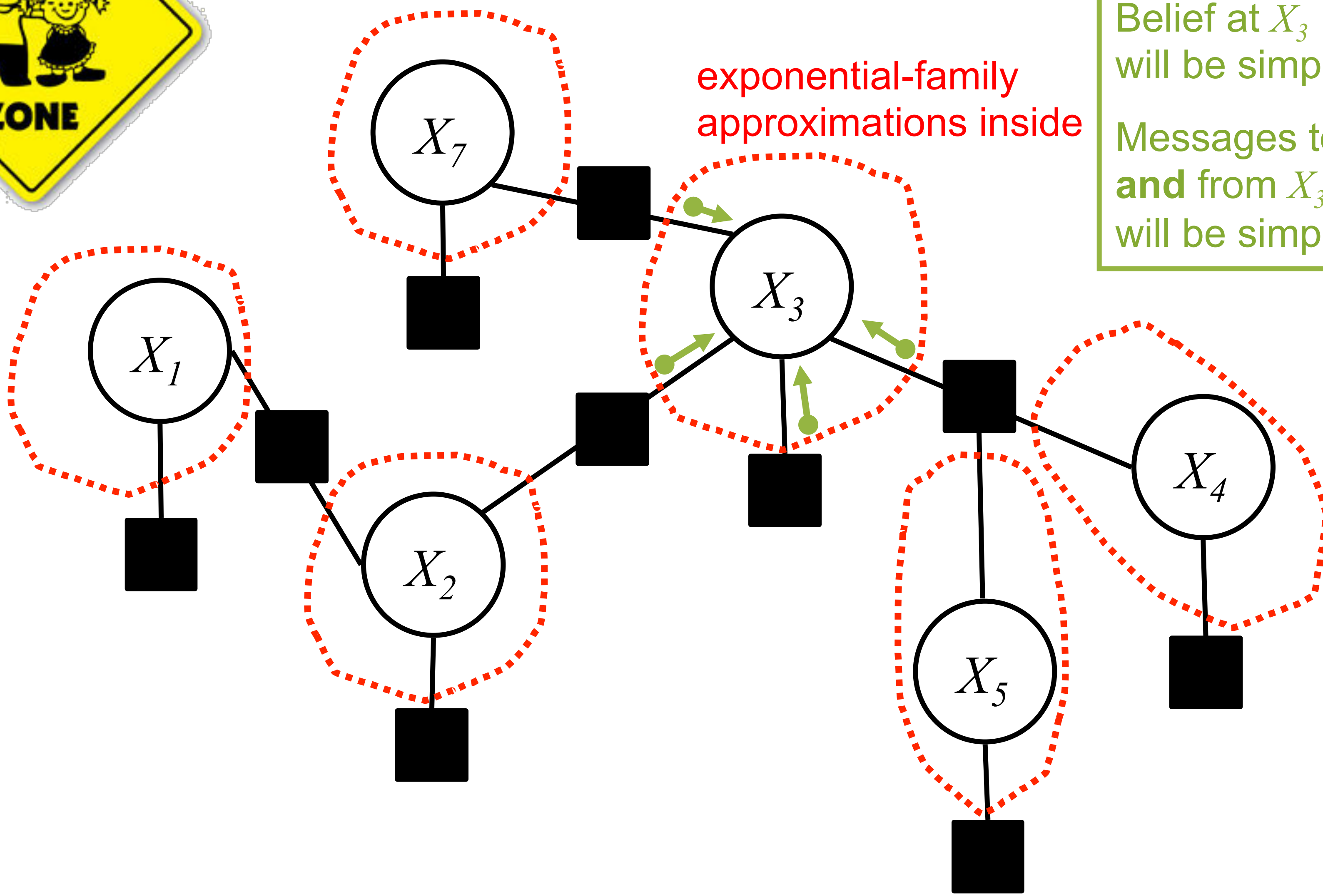
- **Problem:** Product of d messages = complex distribution.
  - **Solution:** Approximate with a simpler distribution.
  - For speed, compute approximation *without* computing full product.

EP provides four special advantages over pruning:

1. **General recipe** that can be used in many settings.
2. **Efficient.** Uses approximations that are very fast.
3. **Conservative.** Unlike pruning, never forces $b(x)$ to 0.
   - Never kills off a value x that had been possible.
4. **Adaptive.** Approximates $\mu(x)$ more carefully if x is favored by the *other* messages.
   - Tries to be accurate on the most "plausible" values.

(Minka, 2001; Heskes & Zoeter, 2002)

Expectation Propagation (EP)



Belief at X_3 will be simple!

Messages to **and** from X_3 will be simple!

Expectation Propagation (EP)

Key idea: Approximate variable X's incoming messages μ.
We force them to have a simple parametric form:

$$\mu(x) = \exp\left(\theta \cdot \mathrm{f}(x)\right)$$ "log-linear model" (unnormalized)

where $\mathrm{f}(x)$ extracts a feature vector from the **value** x.
For each variable X, we'll choose a feature function f.

Maybe unnormalizable, e.g., initial message $\theta=0$ is uniform "distribution"

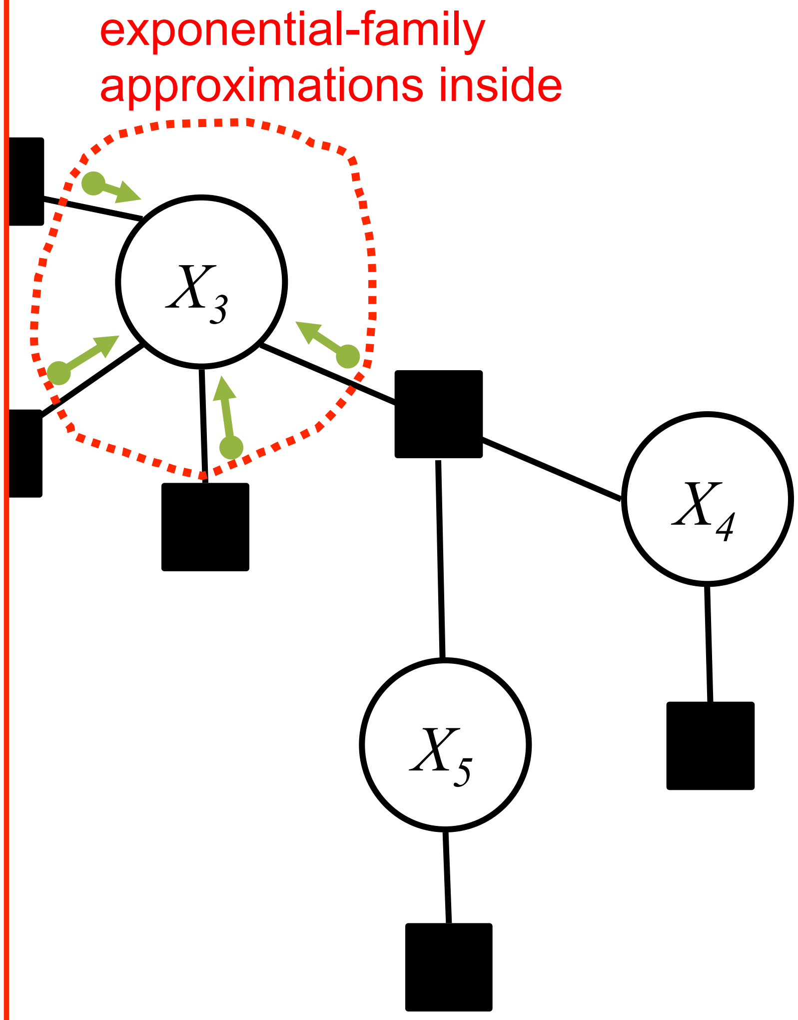
So by storing a few parameters θ, we've defined $\mu(x)$ for all x.
Now the messages are super-easy to multiply:

$$\boldsymbol{\mu}_1(\boldsymbol{x})\, \boldsymbol{\mu}_2(\boldsymbol{x}) = \exp\left(\boldsymbol{\theta} \cdot \mathrm{f}(\boldsymbol{x})\right) \exp\left(\boldsymbol{\theta} \cdot \mathrm{f}(\boldsymbol{x})\right) = \exp\left((\boldsymbol{\theta}_1 + \boldsymbol{\theta}_2) \cdot \boldsymbol{f}(\boldsymbol{x})\right)$$

Represent a message by its parameter vector θ.
To multiply messages, just add their θ vectors!
So beliefs and outgoing messages *also* have this simple form.

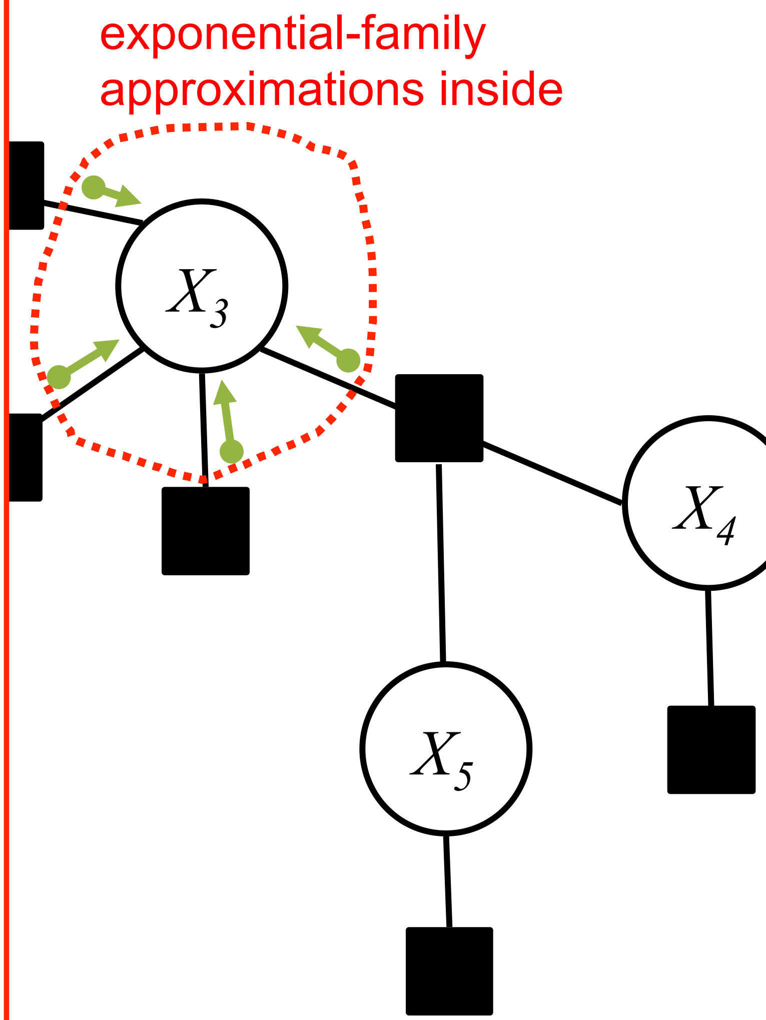
Expectation Propagation

- Form of messages/beliefs at X_3?
  - Always $\mu(x) = \exp(\theta \cdot \mathrm{f}(x))$
- If x is real:
  - Gaussian: Take $\mathrm{f}(x) = (x, x^2)$
- If x is string:
  - Globally normalized trigram model: Take f(x) = (count of aaa, count of aab, … count of zzz)
- If x is discrete:
  - Arbitrary discrete distribution (can exactly represent original message, so we get ordinary BP)
  - Coarsened discrete distribution, based on features of x
- *Can't* use mixture models, or other models that use latent variables to define $\mu(x) = \sum_y p(x, y)$



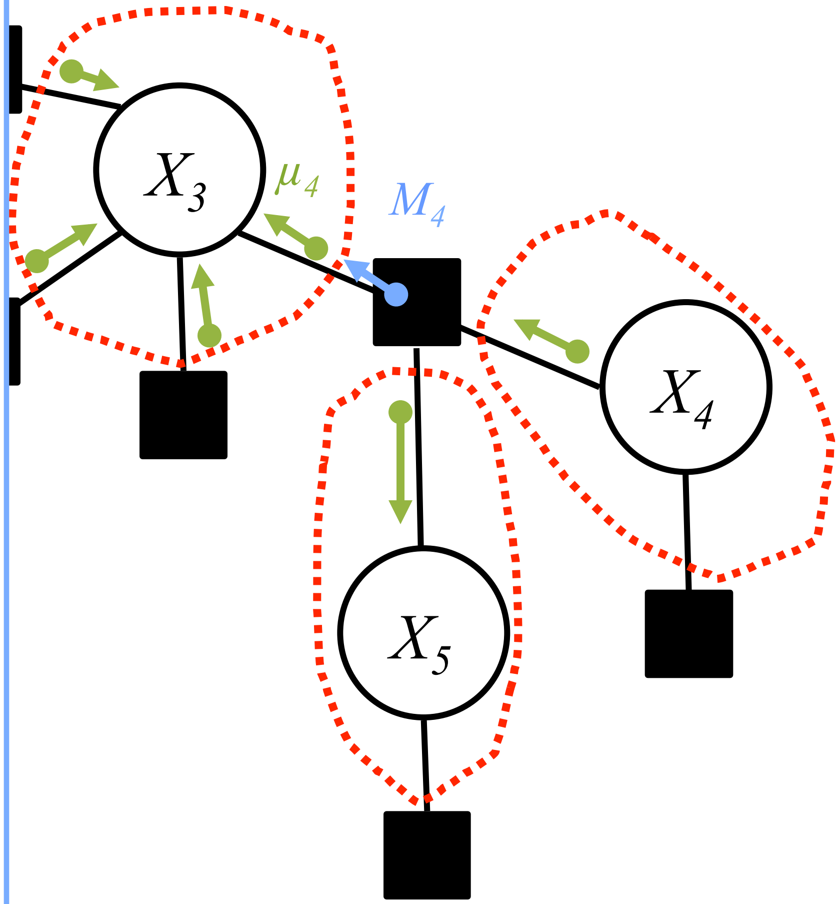
Expectation Propagation

- Each message to X_3 is
  $$\mu(x) = \exp(\theta \cdot f(x))$$
  for some θ. We only store θ.
- To take a product of such messages, just add their θ
  - Easily compute belief at X_3 (sum of incoming θ vectors)
  - Then easily compute each outgoing message (belief minus one incoming θ)
- All very easy …



Expectation Propagation

- But what about messages from factors?
  - Like the message M_4.
  - This is *not* exponential family! Uh-oh!
  - It's just whatever the factor happens to send.
- This is where we need to approximate, by μ_4.



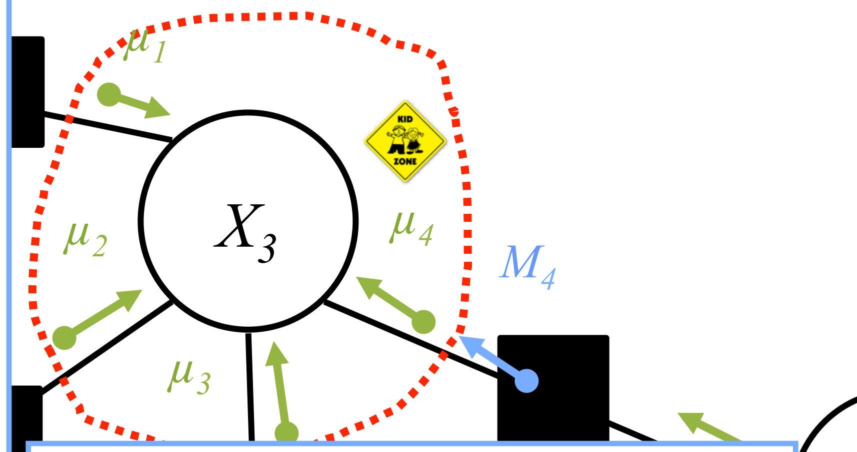
JOHNS HOPKINS UNIVERSITY

Expectation Propagation

- blue = arbitrary distribution,
  green = simple distribution $\exp(\theta \cdot f(x))$

- The **belief** at x "should" be
  $p(x) = \mu_1(x) \cdot \mu_2(x) \cdot \mu_3(x) \cdot M_4(x)$
- But we'll be using
  $b(x) = \mu_1(x) \cdot \mu_2(x) \cdot \mu_3(x) \cdot \mu_4(x)$

- **Choose the simple distribution b that minimizes $KL(p \,||\, b)$.**
- Then, work backward from belief b to message μ_4.
  - Take θ vector of b and subtract off the θ vectors of μ_1, μ_2, μ_3.
- – Chooses μ_4 to preserve *belief* well.

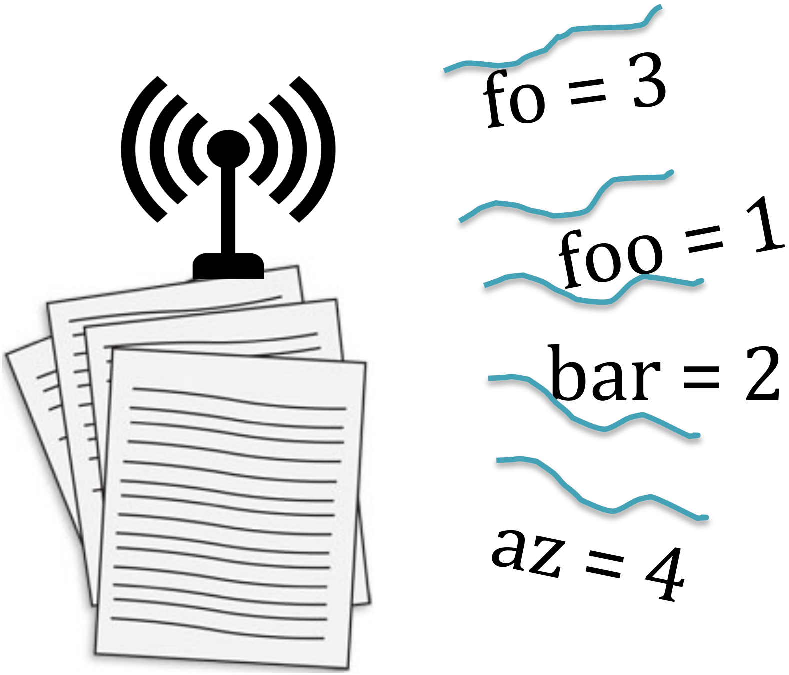


That is, choose b that assigns high probability to samples from p.

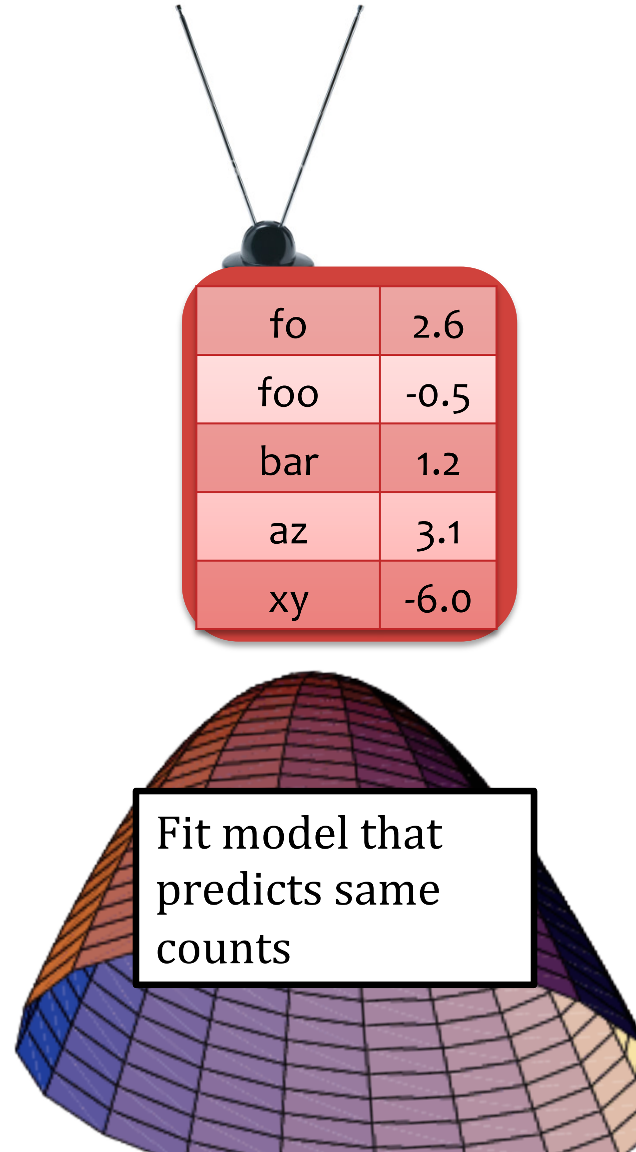
Find b's params θ in closed form – or follow gradient:

$$E_{x\sim p}[f(x)] - E_{x\sim b}[f(x)]$$

ML Estimation = Moment Matching



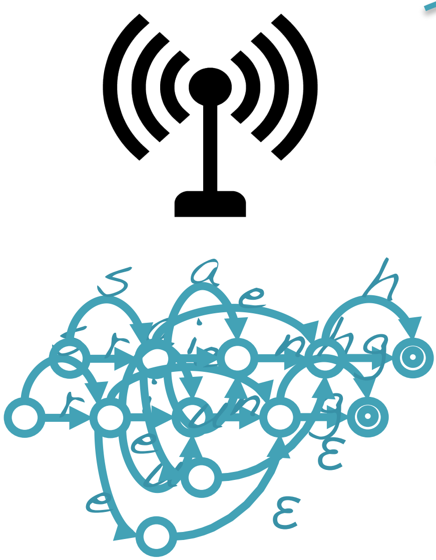
Broadcast *n*-gram counts



fo	2.6
foo	-0.5
bar	1.2
az	3.1
xy	-6.0

Fit model that predicts same counts

FSA Approx. = Moment Matching



fo = 3.1

foo = 0.9

bar = 2.2

zz= 0.1

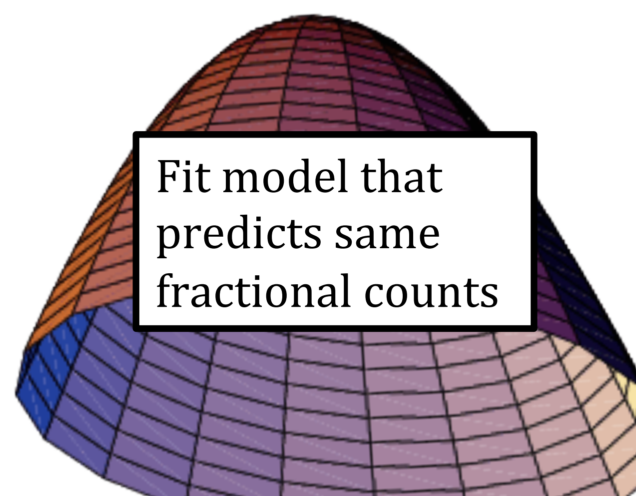
az = 4.1

xx = 0.1

A distribution over strings

(can compute with forward-backward)

fo	2.6
foo	-0.5
bar	1.2
az	3.1
xy	-6.0

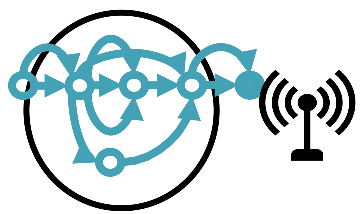


Fit model that predicts same fractional counts

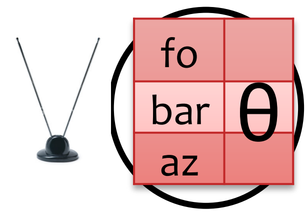
FSA Approx. = Moment Matching

$\min_{\theta}$

KL(



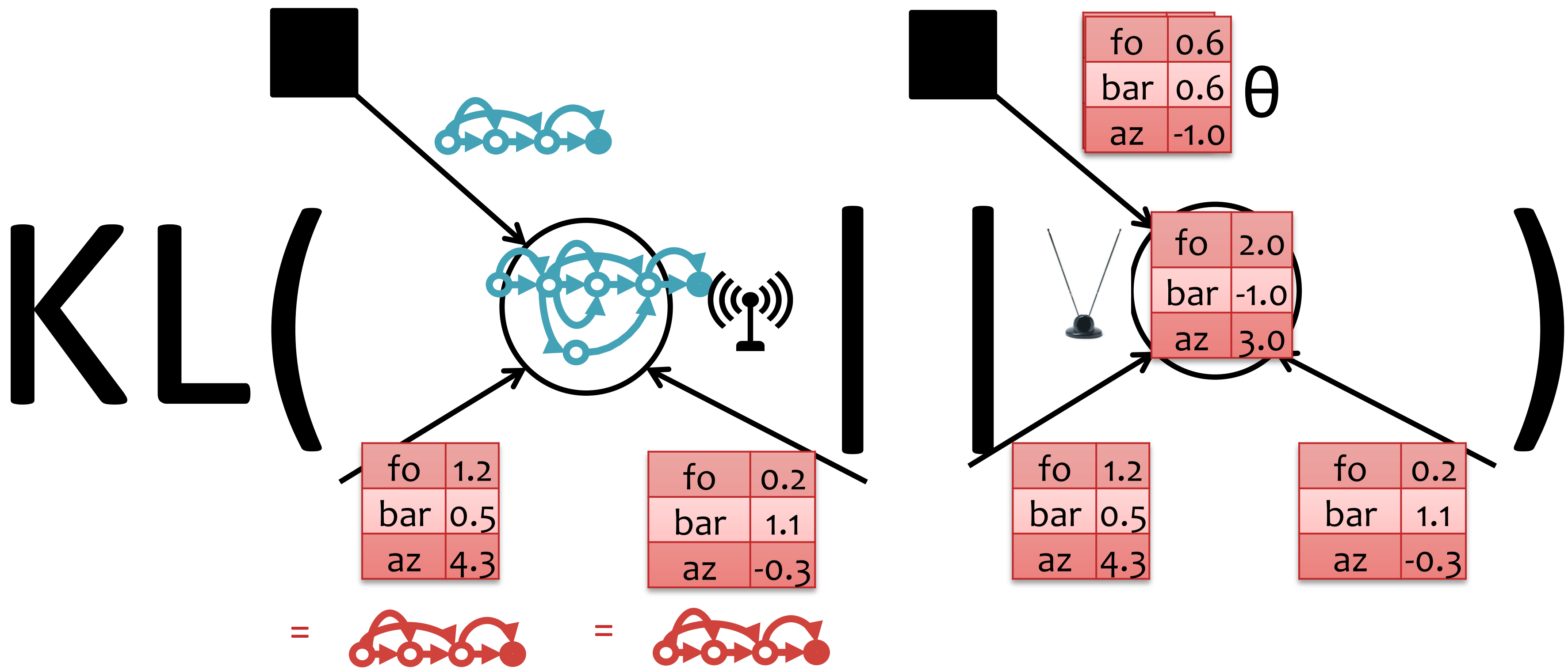
||



)

Finds parameters θ that minimize KL "error" in belief

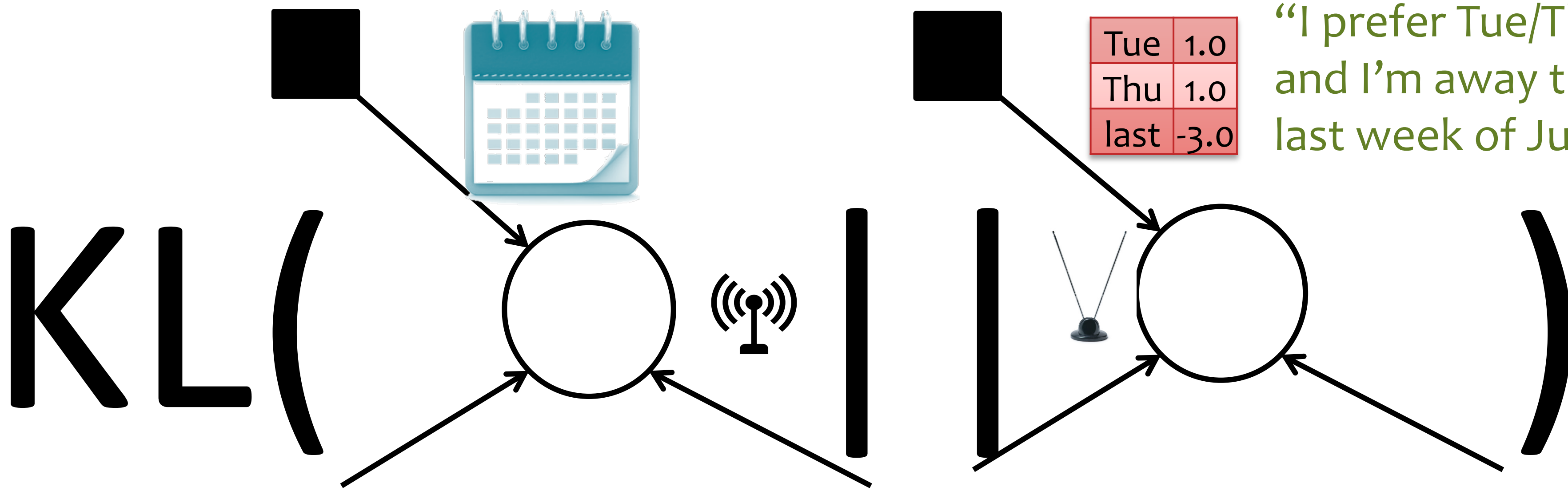
How to approximate a *message*?



Wisely, KL doesn't insist on good approximations for values that are low-probability in the belief

Finds ***message*** parameters θ that minimize KL "error" of resulting ***belief***

Analogy: Scheduling by approximate email messages



Tue	1.0
Thu	1.0
last	-3.0

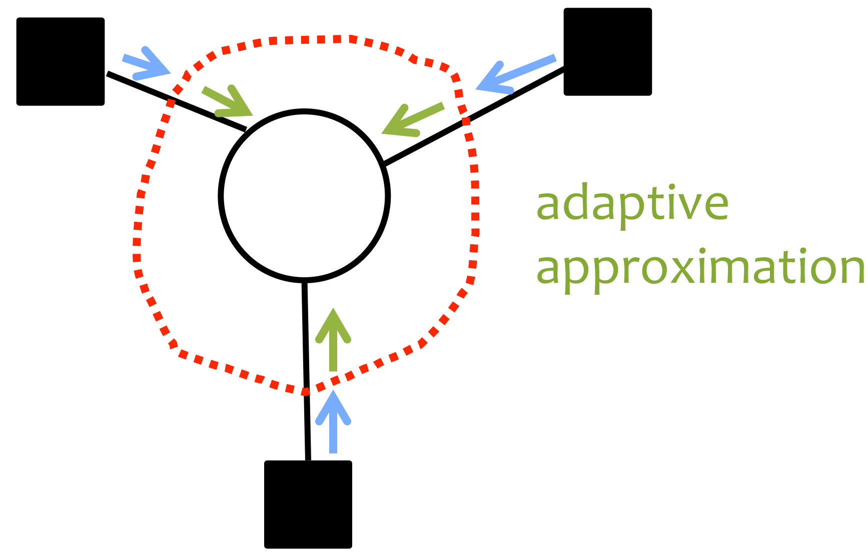
"I prefer Tue/Thu, and I'm away the last week of July."

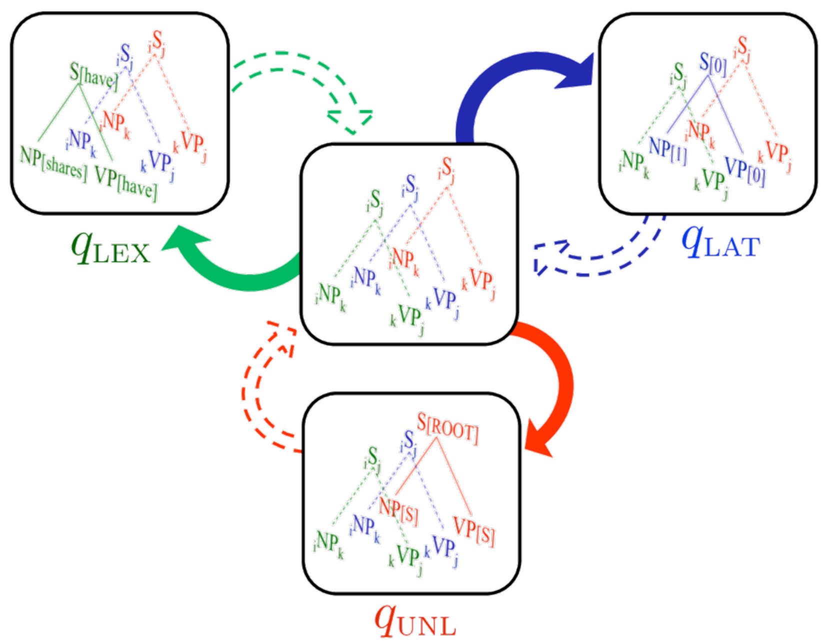
Wisely, KL doesn't insist on good approximations for values that are low-probability in the belief

(This is an approximation to my true schedule. I'm not actually free on all Tue/Thu, but the bad Tue/Thu dates have already been ruled out by messages from other folks.)

Expectation Propagation

- Task: Constituency parsing, with factored annotations
  - Lexical annotations
  - Parent annotations
  - Latent annotations
- Approach:
  - Sentence specific approximation is an anchored grammar: $q(A \rightarrow B\ C,\ i,\ j,\ k)$
  - Sending messages is equivalent to marginalizing out the annotations





(Hall & Klein, 2012)

Section 6: Approximation-aware Training

Outline

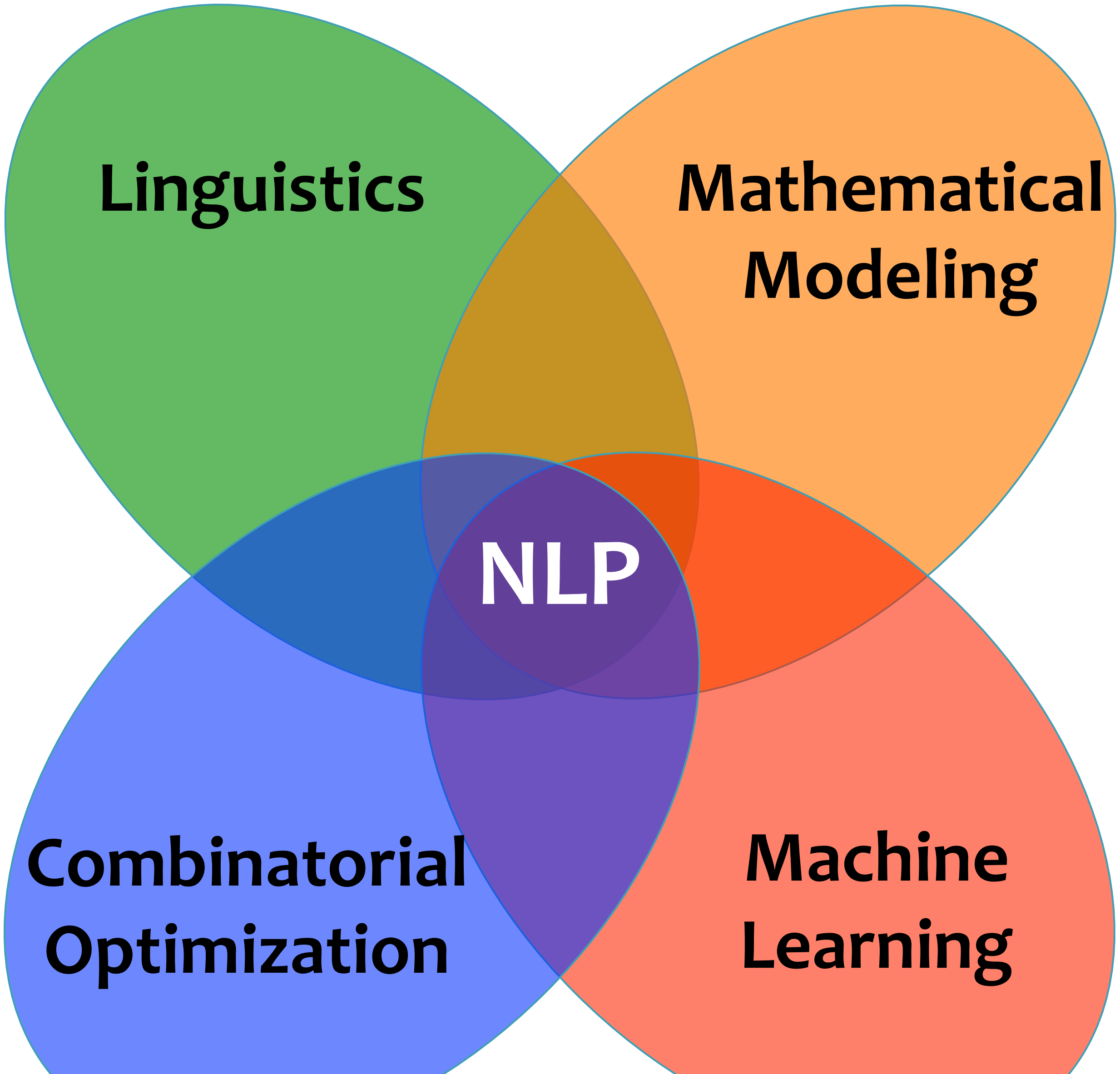
- Do you want to push past the simple NLP models (logistic regression, PCFG, etc.) that we've all been using for 20 years?
- Then this tutorial is extremely practical for you!
  1. **Models:** Factor graphs can express interactions among linguistic structures.
  2. **Algorithm:** BP estimates the global effect of these interactions on each variable, using local computations.
  3. **Intuitions:** What's going on here? Can we trust BP's estimates?
  4. **Fancier Models:** Hide a whole grammar and dynamic programming algorithm within a single factor. BP coordinates multiple factors.
  5. **Tweaked Algorithm:** Finish in fewer steps and make the steps faster.
  6. **Learning:** Tune the parameters. Approximately improve the true predictions -- or truly improve the approximate predictions.
  7. **Software:** Build the model you want!

Outline

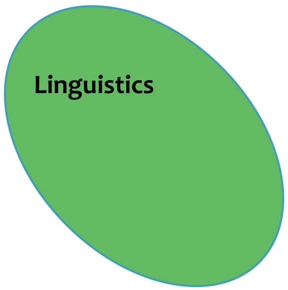
- Do you want to push past the simple NLP models (logistic regression, PCFG, etc.) that we've all been using for 20 years?
- Then this tutorial is extremely practical for you!

1. **Models:** Factor graphs can express interactions among linguistic structures.
2. **Algorithm:** BP estimates the global effect of these interactions on each variable, using local computations.
3. **Intuitions:** What's going on here? Can we trust BP's estimates?
4. **Fancier Models:** Hide a whole grammar and dynamic programming algorithm within a single factor. BP coordinates multiple factors.
5. **Tweaked Algorithm:** Finish in fewer steps and make the steps faster.
6. **Learning:** Tune the parameters. Approximately improve the true predictions -- or truly improve the approximate predictions.
7. **Software:** Build the model you want!

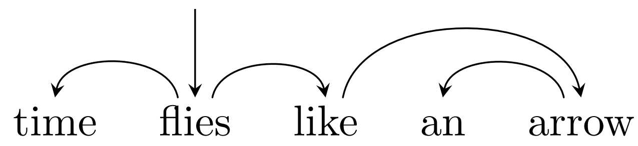
Modern NLP

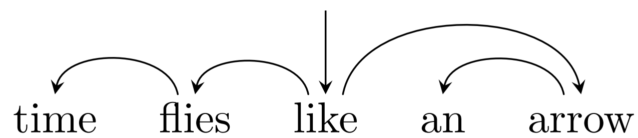

Linguistics
Mathematical Modeling
NLP
Combinatorial Optimization
Machine Learning

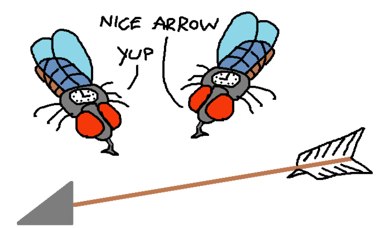
Machine Learning for NLP

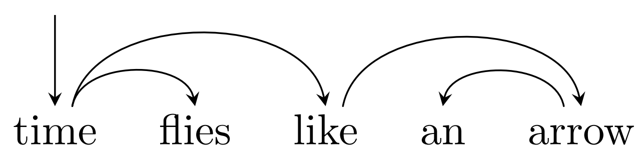


Linguistics inspires the structures we want to predict

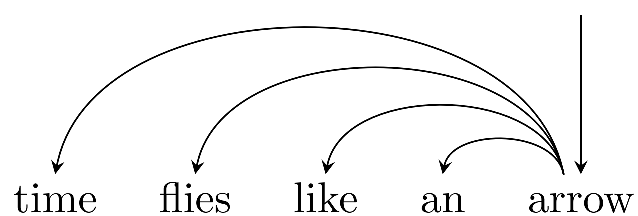






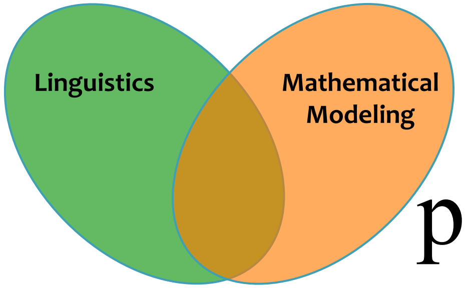


. . .

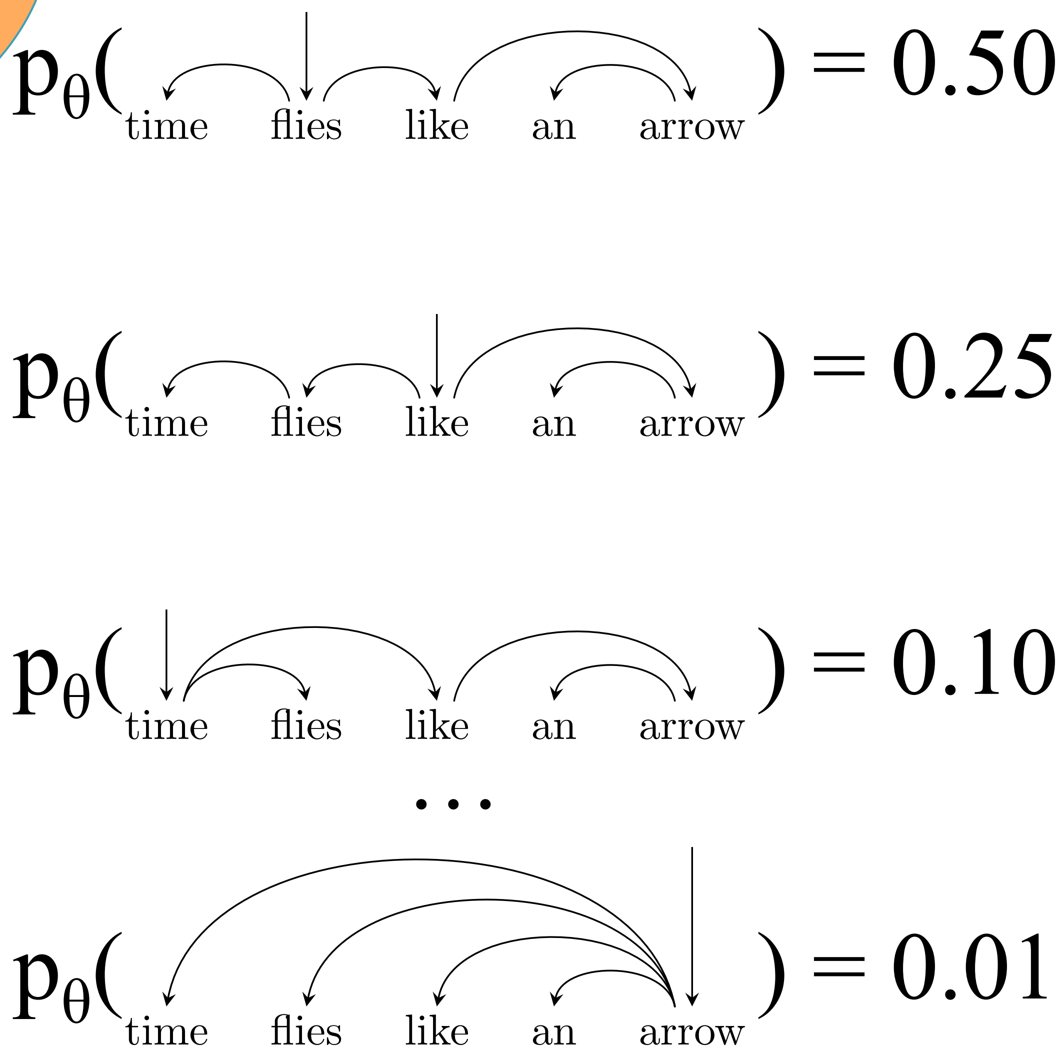


No semantic interpretation

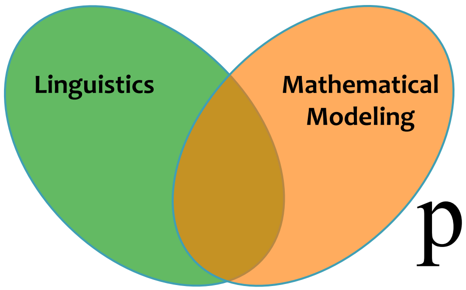
Machine Learning for NLP



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



Machine Learning for NLP



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

It also tells us what to optimize

$$p_\theta(\text{time flies like an arrow}) = 0.50$$

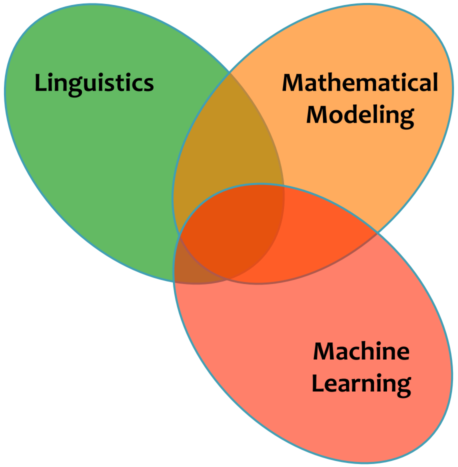
$$p_\theta(\text{time flies like an arrow}) = 0.25$$

$$p_\theta(\text{time flies like an arrow}) = 0.10$$

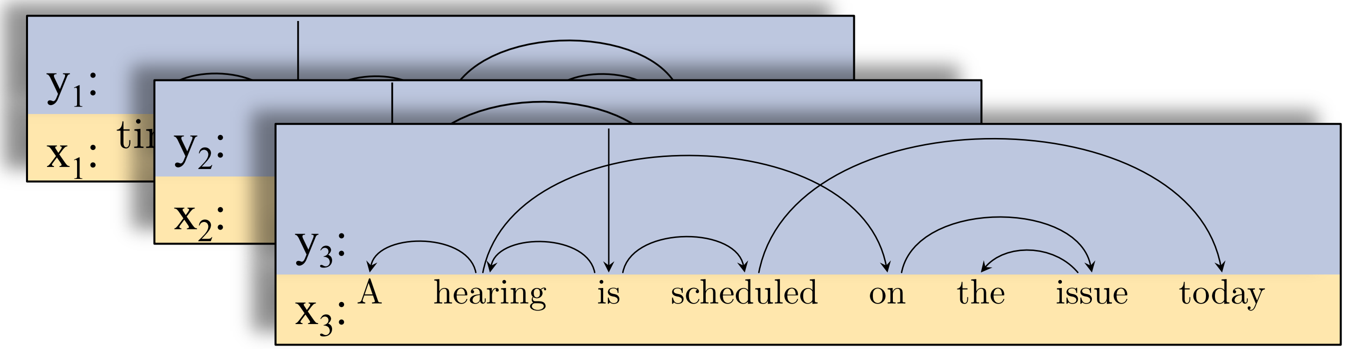
$$\ldots$$

$$p_\theta(\text{time flies like an arrow}) = 0.01$$

Machine Learning for NLP



Given training instances $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$

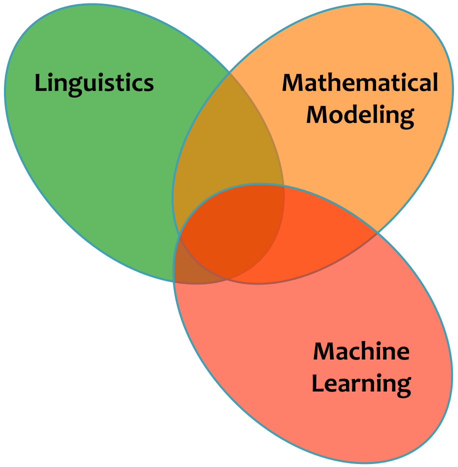


Learning tunes the parameters of the model

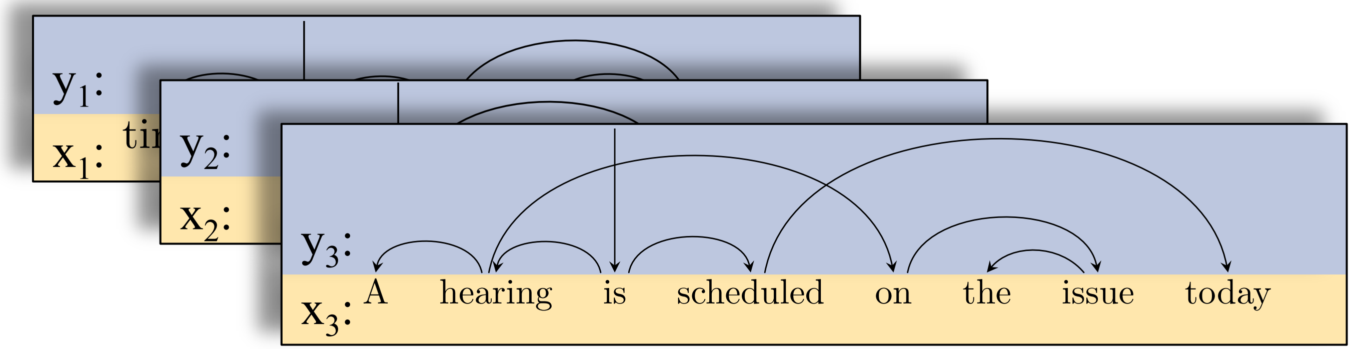
Find the best model parameters, $\boldsymbol{\theta}$

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \prod_{i=1}^{n} p_{\boldsymbol{\theta}}(\boldsymbol{y}_i \mid \boldsymbol{x}_i)$$

Machine Learning for NLP

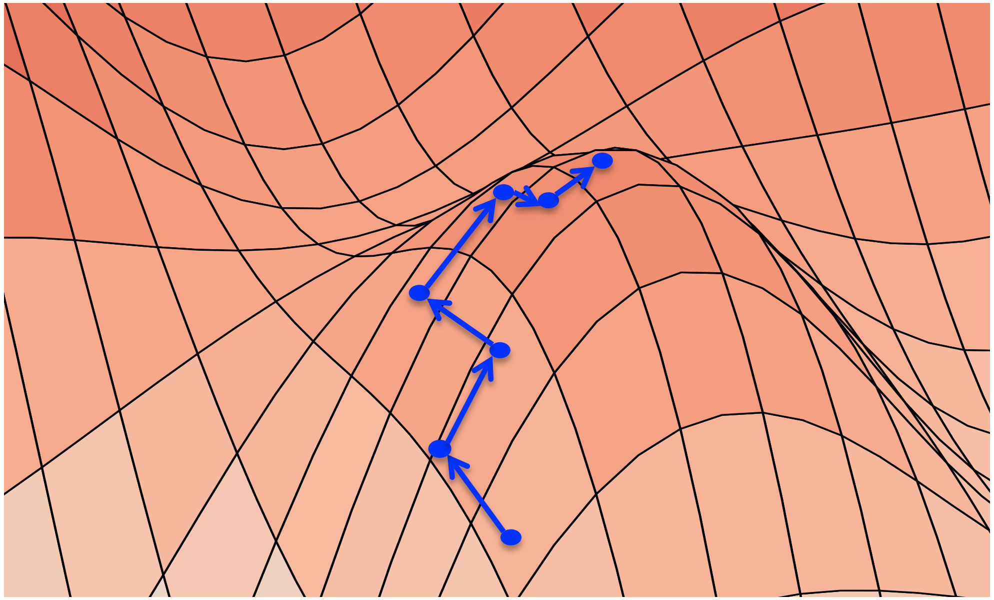


Given training instances $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$

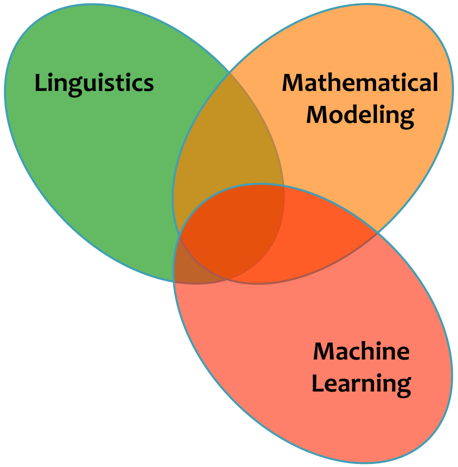


Learning tunes the parameters of the model

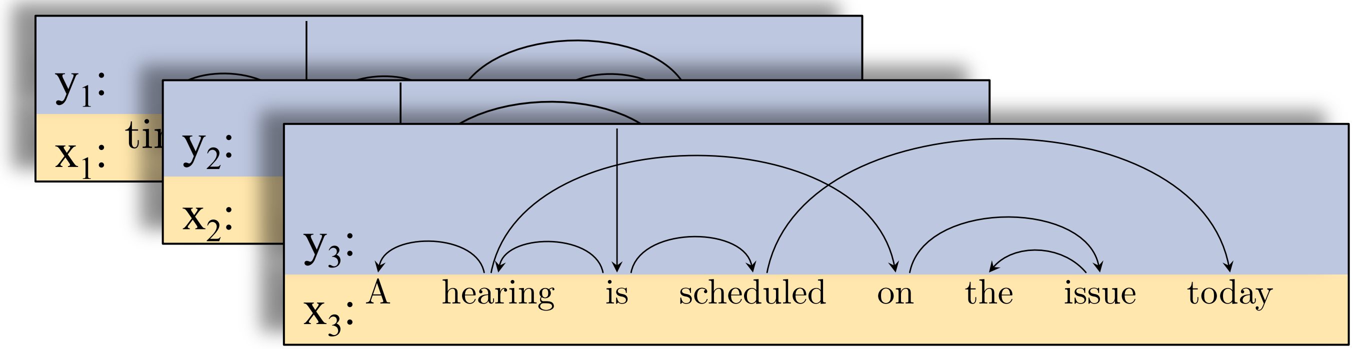
Find the best model parameters, $\boldsymbol{\theta}$



Machine Learning for NLP

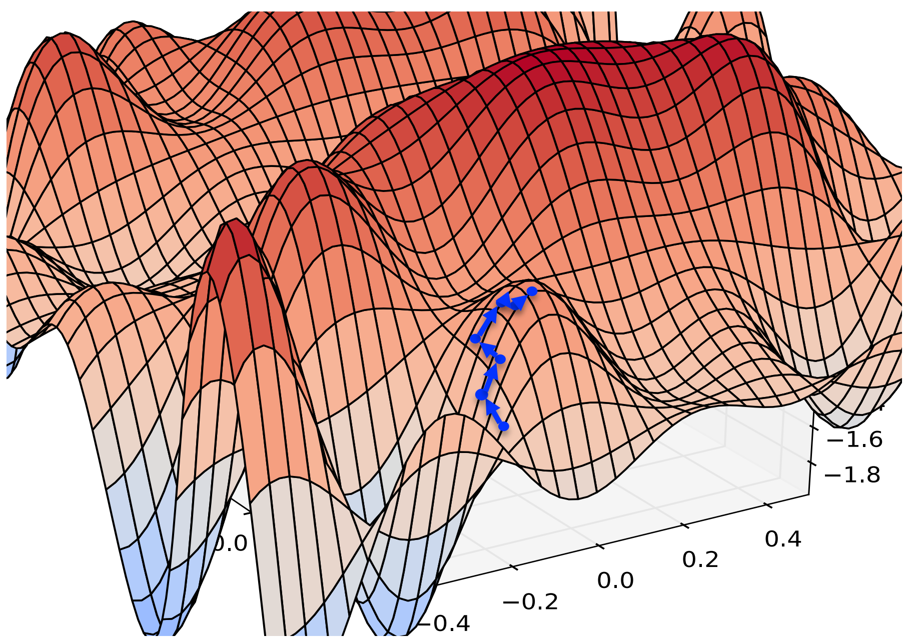


Given training instances $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$

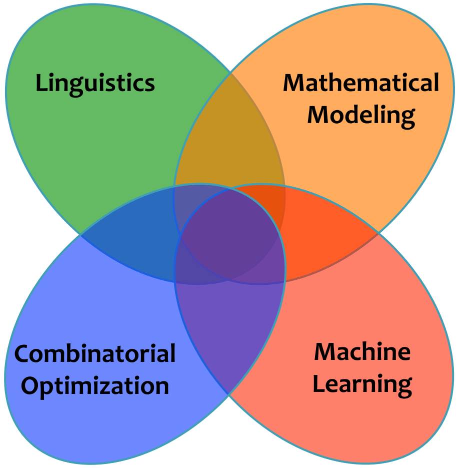


Find the best model parameters, $\boldsymbol{\theta}$

Learning tunes the parameters of the model

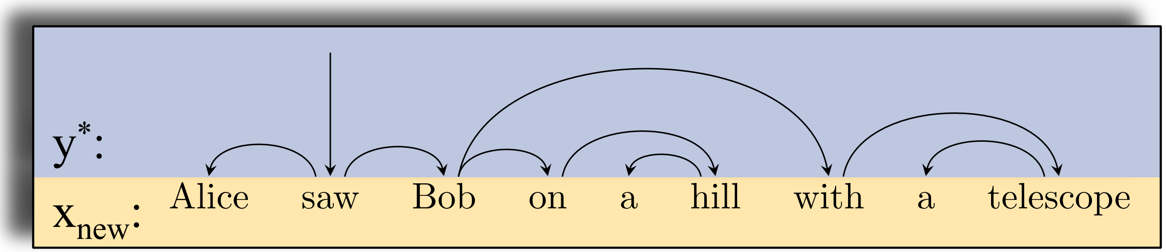


Machine Learning for NLP



- Given a **new sentence,** x_{new}
- Search over the **set of all possible structures** (often exponential in size of x_{new})
- Return the **highest scoring** structure, y^*

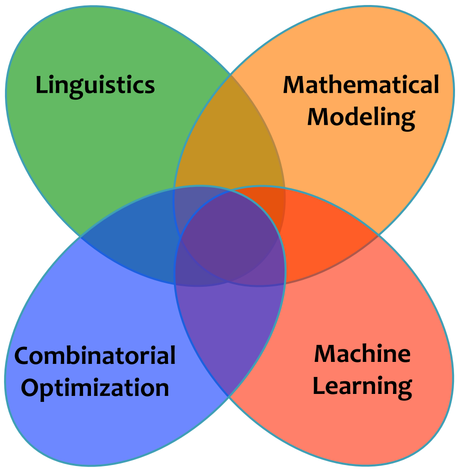
$$\boldsymbol{y}^* = \underset{\boldsymbol{y}}{\operatorname{argmax}}\, p_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x}_{\text{new}})$$



Inference finds the best structure for a new sentence

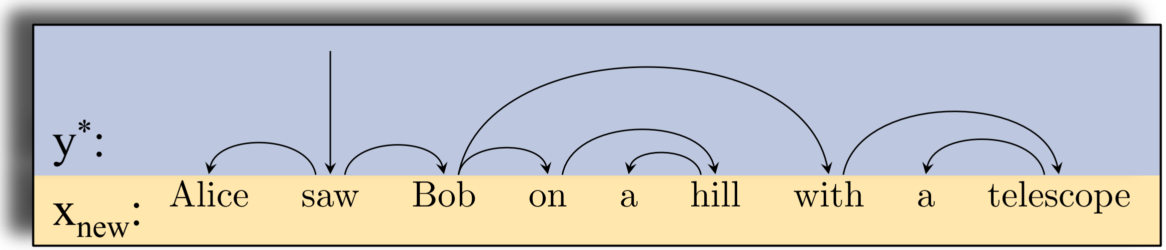
(Inference is usually called as a **subroutine** in learning)

Machine Learning for NLP



- Given a **new sentence,** x_{new}
- Search over the **set of all possible structures** (often exponential in size of x_{new})
- Return the **Minimum Bayes Risk (MBR)** structure, y^*

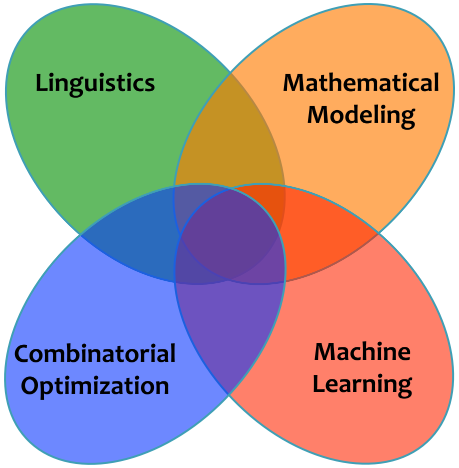
$$\boldsymbol{y}^* = \operatorname*{argmin}_{\boldsymbol{y}} \mathbb{E}_{p_\theta(\boldsymbol{y}'|\boldsymbol{x})}[\ell(\boldsymbol{y}, \boldsymbol{y}')]$$



Inference finds the best structure for a new sentence

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

Machine Learning for NLP

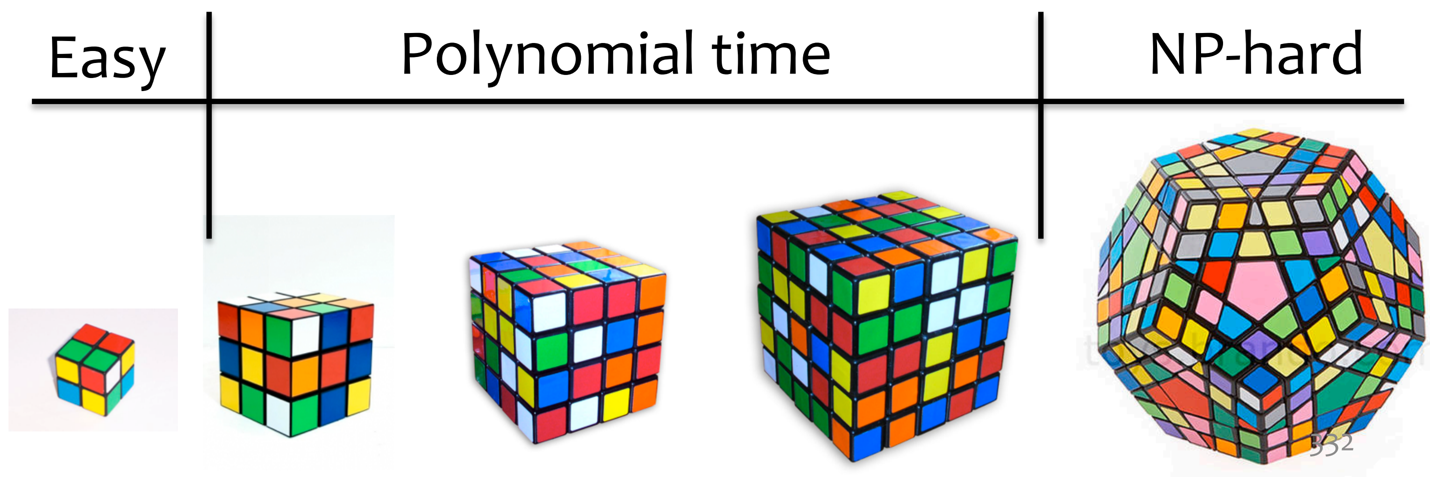


- Given a **new sentence,** x_{new}
- Search over the **set of all possible structures** (often exponential in size of x_{new})
- Return the **Minimum Bayes Risk (MBR)** structure, y^*

$$\boldsymbol{y}^* = \operatorname*{argmin}_{\boldsymbol{y}} \mathbb{E}_{p_\theta(\boldsymbol{y}'|\boldsymbol{x})}[\ell(\boldsymbol{y}, \boldsymbol{y}')]$$

Inference finds the best structure for a new sentence

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

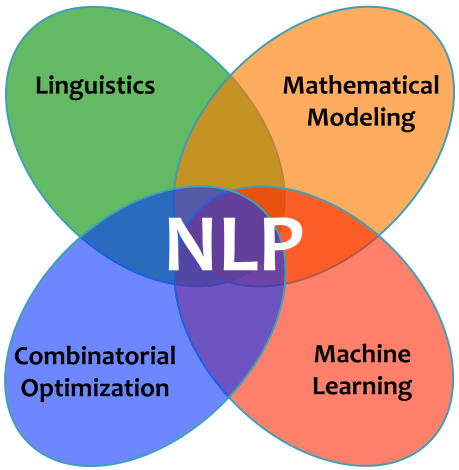


Modern NLP

Linguistics inspires the structures we want to predict

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

It also tells us what to optimize



Learning tunes the parameters of the model

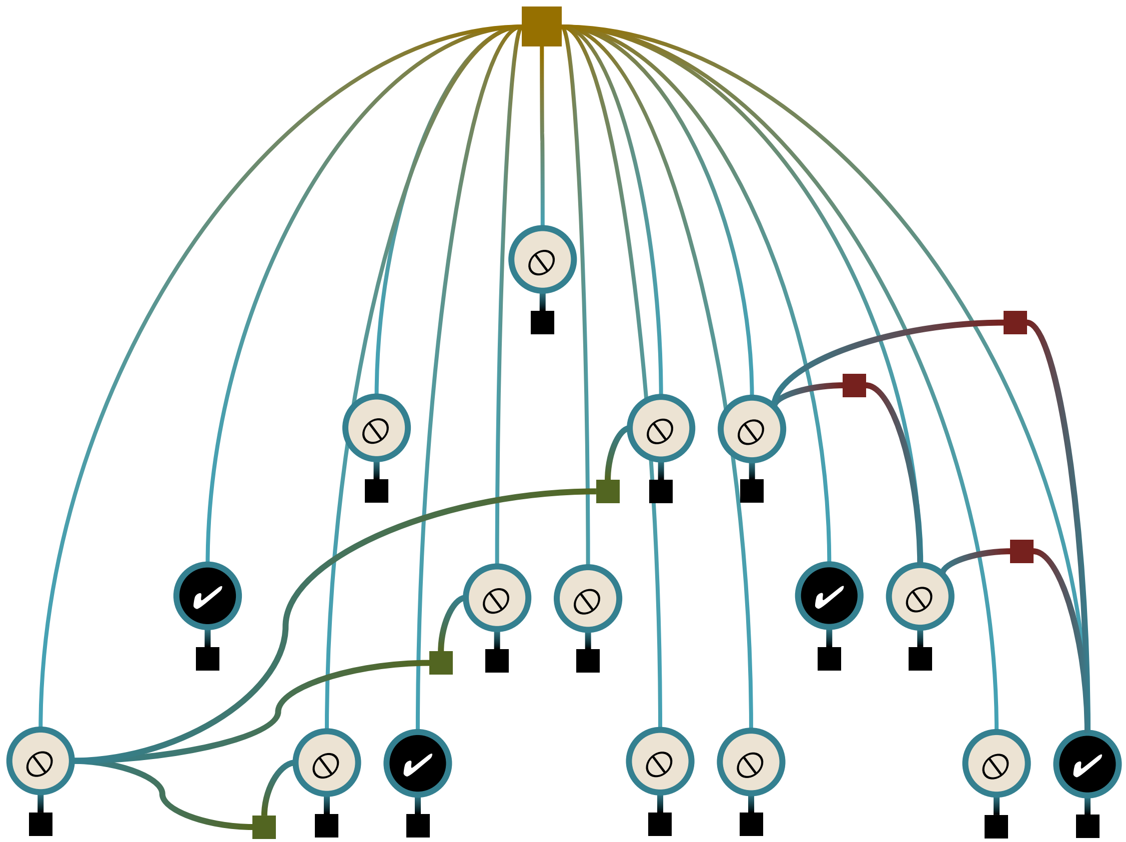
Inference finds the best structure for a new sentence

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

Mathematical Modeling

An Abstraction for Modeling

Now we can work at this level of abstraction.



$$p_\theta(\boldsymbol{y}) = \frac{1}{Z} \prod_\alpha \psi_\alpha(\boldsymbol{y}_\alpha)$$

Training

Thus far, we've seen how to compute (approximate) marginals, given a factor graph…

…but where do the potential tables ψ_α come from?

- Some have a fixed structure (e.g. *Exactly1, CKYTree*)
- Others could be trained ahead of time (e.g. *TrigramHMM*)
- For the rest, we **define them parametrically and learn the parameters!**

Two ways to learn:

1. **Standard CRF Training** (very simple; often yields state-of-the-art results)
2. **ERMA** (less simple; but takes approximations and loss function into account)

Standard CRF Parameterization

Define each potential function in terms of a fixed set of feature functions:

$$\psi_\alpha(\boldsymbol{x}_\alpha, \boldsymbol{y}_\alpha; \boldsymbol{\theta}) = \exp(\boldsymbol{\theta} \cdot \boldsymbol{f}_\alpha(\boldsymbol{x}_\alpha, \boldsymbol{y}_\alpha))$$

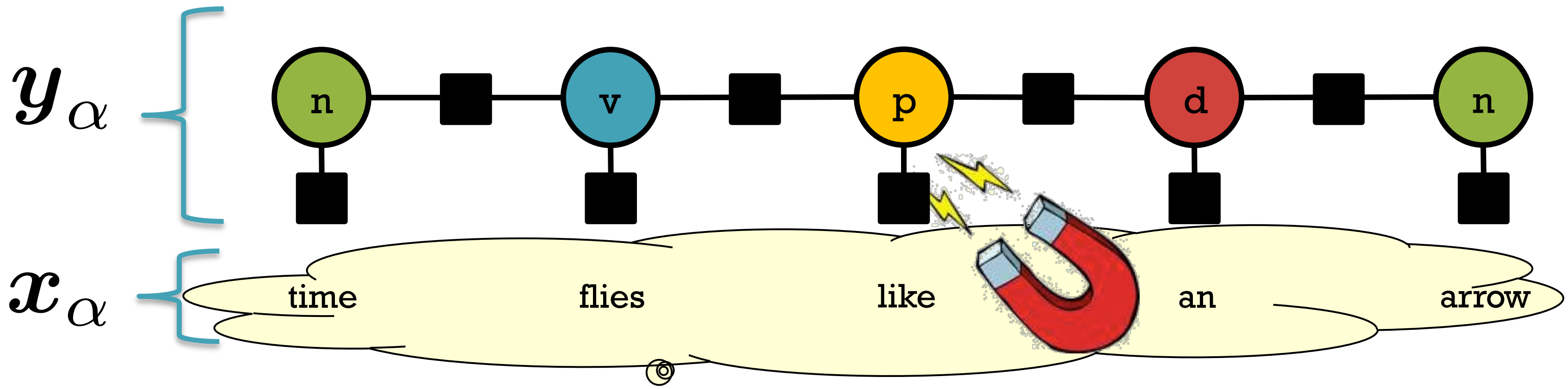
Observed variables

Predicted variables

Standard CRF Parameterization

Define each potential function in terms of a fixed set of feature functions:

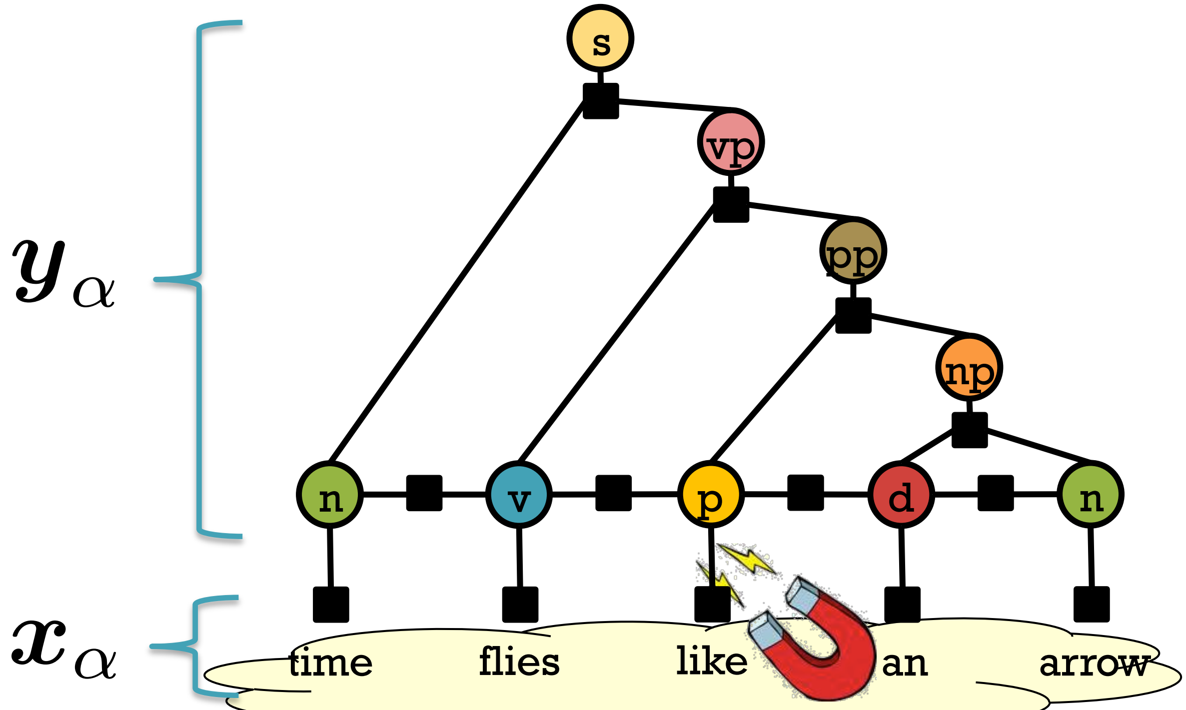
$$\psi_\alpha(\boldsymbol{x}_\alpha, \boldsymbol{y}_\alpha; \boldsymbol{\theta}) = \exp(\boldsymbol{\theta} \cdot \boldsymbol{f}_\alpha(\boldsymbol{x}_\alpha, \boldsymbol{y}_\alpha))$$



Standard CRF Parameterization

Define each potential function in terms of a fixed set of feature functions:

$$\psi_\alpha(\boldsymbol{x}_\alpha, \boldsymbol{y}_\alpha; \boldsymbol{\theta}) = \exp(\boldsymbol{\theta} \cdot \boldsymbol{f}_\alpha(\boldsymbol{x}_\alpha, \boldsymbol{y}_\alpha))$$



What is Training?

That’s easy:

Training = picking **good** model parameters!

But how do we know if the
model parameters are any “good”?

Machine Learning

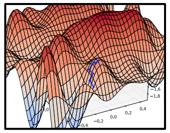
Conditional Log-likelihood Training

1. Choose **model**

$$p_\theta(\boldsymbol{y}) = \frac{1}{Z} \prod_\alpha \psi_\alpha(\boldsymbol{y}_\alpha)$$

2. Choose **objective:** Assign high probability to the things we observe and low probability to everything else

$$L(\theta) = \sum_{\boldsymbol{y} \in \mathcal{D}} \log p_\theta(\boldsymbol{y})$$



3. Compute derivative **by hand** using the chain rule

$$\frac{dL(\theta)}{d\theta_j} = \sum_{\boldsymbol{y} \in \mathcal{D}} \left(\sum_\alpha \left[f_{\alpha,j}(\boldsymbol{y}_\alpha) - \sum_{\boldsymbol{y}'} p_\theta(\boldsymbol{y}'_\alpha) f_{\alpha,j}(\boldsymbol{y}'_\alpha) \right] \right)$$

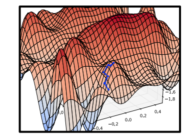
Conditional Log-likelihood Training

1. Choose **model**
   Such that derivative in #3 is easy

$$p_\theta(\boldsymbol{y}) = \frac{1}{Z} \prod_\alpha \exp(\theta \cdot \boldsymbol{f}_\alpha(\boldsymbol{y}_\alpha))$$

2. Choose **objective:**
   Assign high probability to the things we observe and low probability to everything else

$$L(\theta) = \sum_{\boldsymbol{y} \in \mathcal{D}} \log p_\theta(\boldsymbol{y})$$



3. Compute derivative **by hand** using the chain rule

$$\frac{dL(\theta)}{d\theta_j} = \sum_{\boldsymbol{y} \in \mathcal{D}} \left(\sum_\alpha \left[f_{\alpha,j}(\boldsymbol{y}_\alpha) - \sum_{\boldsymbol{y}'} \boxed{p_\theta(\boldsymbol{y}'_\alpha)} f_{\alpha,j}(\boldsymbol{y}'_\alpha) \right] \right)$$

4. Replace **exact inference** by **approximate inference**

$$\boxed{\approx} \sum_{\boldsymbol{y} \in \mathcal{D}} \left(\sum_\alpha \left[f_{\alpha,j}(\boldsymbol{y}_\alpha) - \sum_{\boldsymbol{y}'} \boxed{b_\theta(\boldsymbol{y}'_\alpha)} f_{\alpha,j}(\boldsymbol{y}'_\alpha) \right] \right)$$

We can **approximate** the **factor marginals** by the (normalized) **factor beliefs** from BP!

Stochastic Gradient Descent

Input:

– Training data, $\{(x^{(i)}, y^{(i)}) : 1 \leq i \leq N\}$

– Initial model parameters, θ

Output:

– Trained model parameters, θ.

Algorithm:

While not converged:

– Sample a training example $(x^{(i)}, y^{(i)})$

– Compute the gradient of $\log(p_\theta(y^{(i)} \mid x^{(i)}))$ with respect to our model parameters θ.

– Take a (small) step in the direction of the gradient.

What's wrong with the usual approach?

If you add too many factors, your predictions might get worse!

- The model might be richer, but we replace the **true marginals** with **approximate marginals** (e.g. beliefs computed by BP)
- Approximate inference can cause gradients for structured learning to go **awry**! (Kulesza & Pereira, 2008).

What's wrong with the usual approach?

Mistakes made by Standard CRF Training:

1. Using BP (approximate)
2. Not taking loss function into account
3. Should be doing MBR decoding

Big pile of approximations…

…which has tunable parameters.

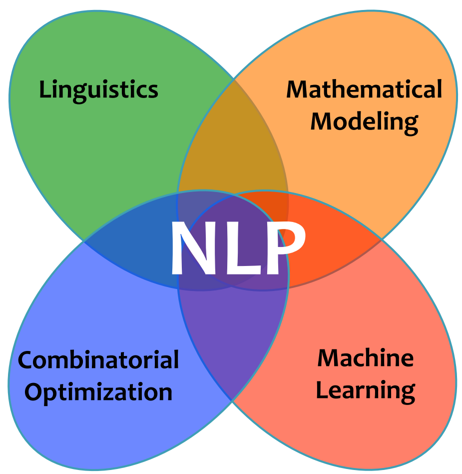
Treat it like a neural net, and run backprop!

Modern NLP

Linguistics inspires the structures we want to predict

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

It also tells us what to optimize



Learning tunes the parameters of the model

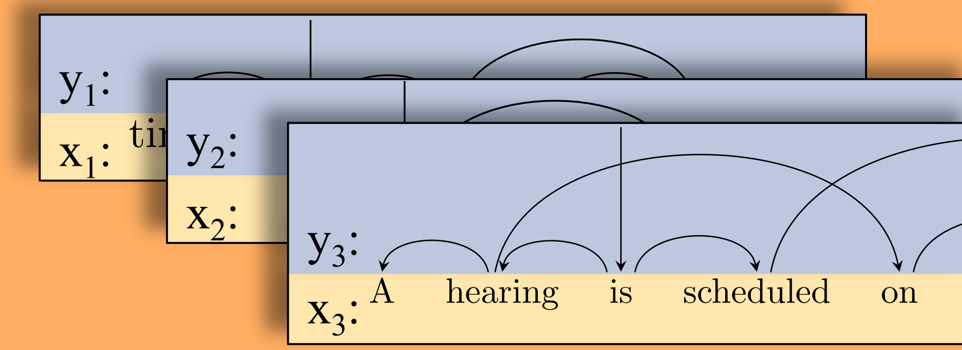
Inference finds the best structure for a new sentence

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

Empirical Risk Minimization

1. Given training data:

$$\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$$



2. Choose each of these:

– Decision function

$$\hat{\boldsymbol{y}} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$

Examples: Linear regression, Logistic regression, Neural Network

– Loss function

$$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R}$$

Examples: Mean-squared error, Cross Entropy

Empirical Risk Minimization

1. Given training data:

$$\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$$

2. Choose each of these:

– Decision function

$$\hat{\boldsymbol{y}} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$

– Loss function

$$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R}$$

3. Define goal:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^N \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

4. Train with SGD:

(take small steps opposite the gradient)

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

Empirical Risk Minimization

1. Given training data:

$$\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$$

2. Choose each of these:

– Decision function

$$\hat{y} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$

– Loss function

$$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R}$$

3. Define goal:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^N \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

4. Train with SGD:

(take small steps opposite the gradient)

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

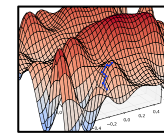
Conditional Log-likelihood Training

1. Choose **model**
   Such that derivative in #3 is easy

$$p_\theta(\boldsymbol{y}) = \frac{1}{Z} \prod_{\alpha} \exp(\theta \cdot \boldsymbol{f}_\alpha(\boldsymbol{y}_\alpha))$$

2. Choose **objective:**
   Assign high probability to the things we observe and low probability to everything else

$$L(\theta) = \sum_{\boldsymbol{y} \in \mathcal{D}} \log p_\theta(\boldsymbol{y})$$



3. Compute derivative **by hand** using the chain rule

$$\frac{dL(\theta)}{d\theta_j} = \sum_{\boldsymbol{y} \in \mathcal{D}} \left(\sum_{\alpha} \left[f_{\alpha,j}(\boldsymbol{y}_\alpha) - \sum_{\boldsymbol{y}'} \boxed{p_\theta(\boldsymbol{y}'_\alpha)} f_{\alpha,j}(\boldsymbol{y}'_\alpha) \right] \right)$$

4. Replace **true inference** by **approximate inference**

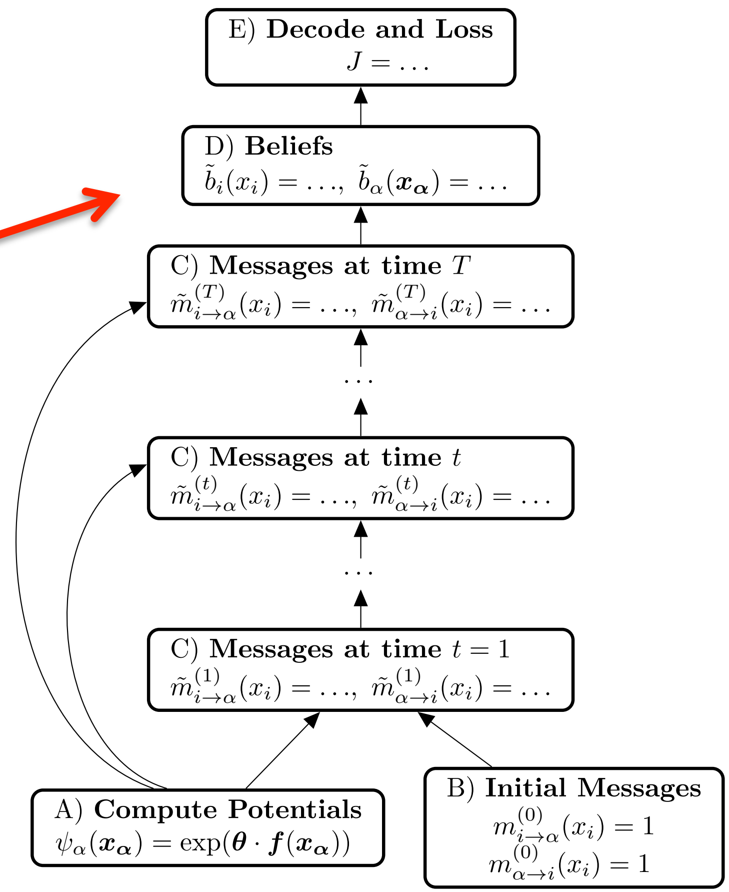
$$\boxed{\approx} \sum_{\boldsymbol{y} \in \mathcal{D}} \left(\sum_{\alpha} \left[f_{\alpha,j}(\boldsymbol{y}_\alpha) - \sum_{\boldsymbol{y}'} \boxed{b_\theta(\boldsymbol{y}'_\alpha)} f_{\alpha,j}(\boldsymbol{y}'_\alpha) \right] \right)$$

What went wrong?

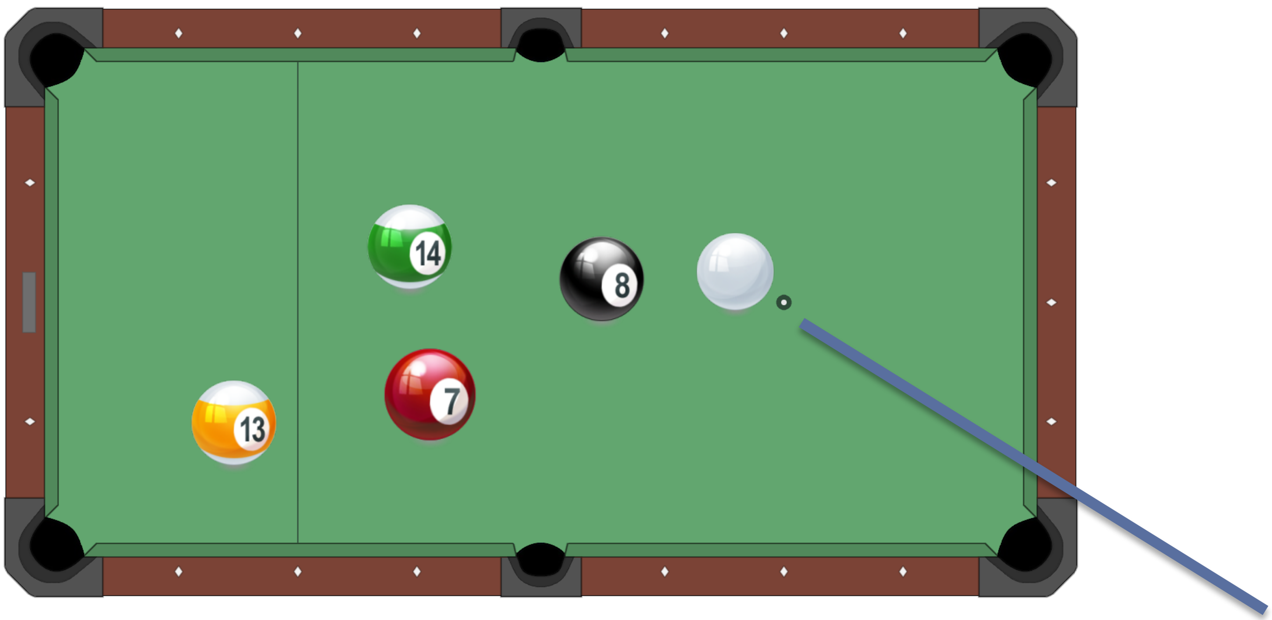
How did we compute these **approximate** marginal probabilities anyway?

By Belief Propagation of course!

$$b_\theta(\boldsymbol{y}'_\alpha)$$

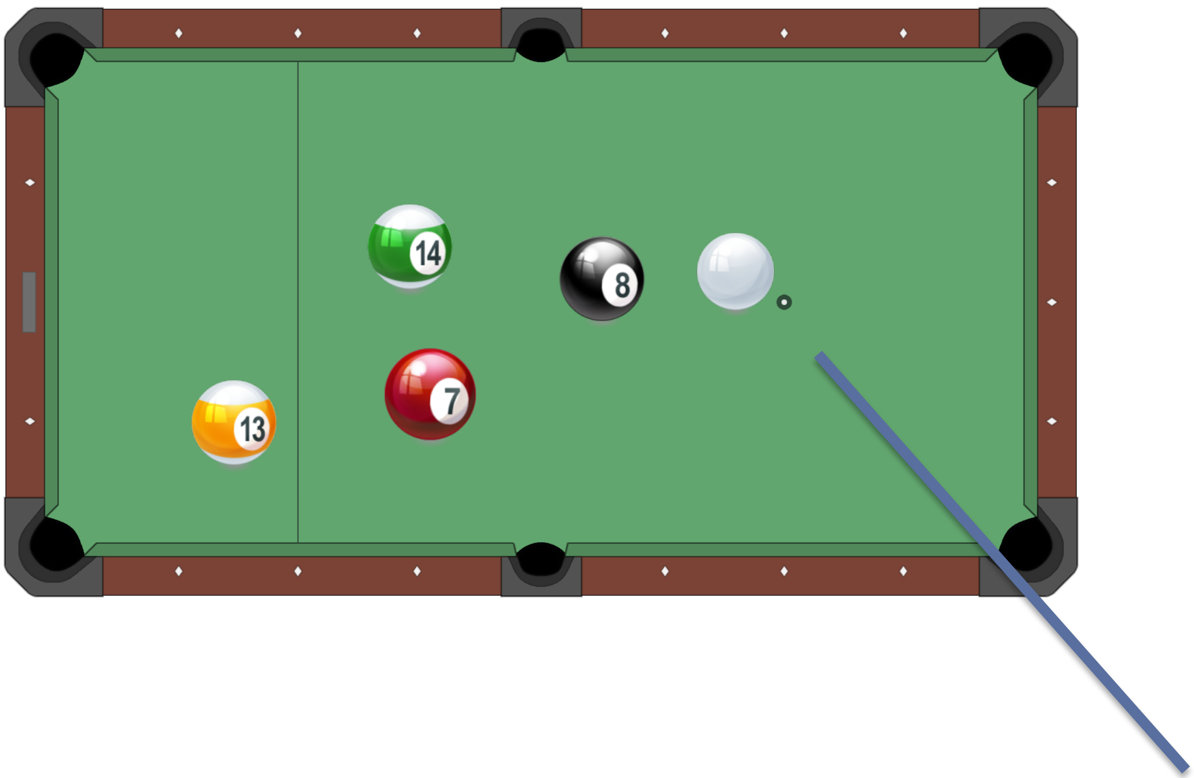


Error Back-Propagation



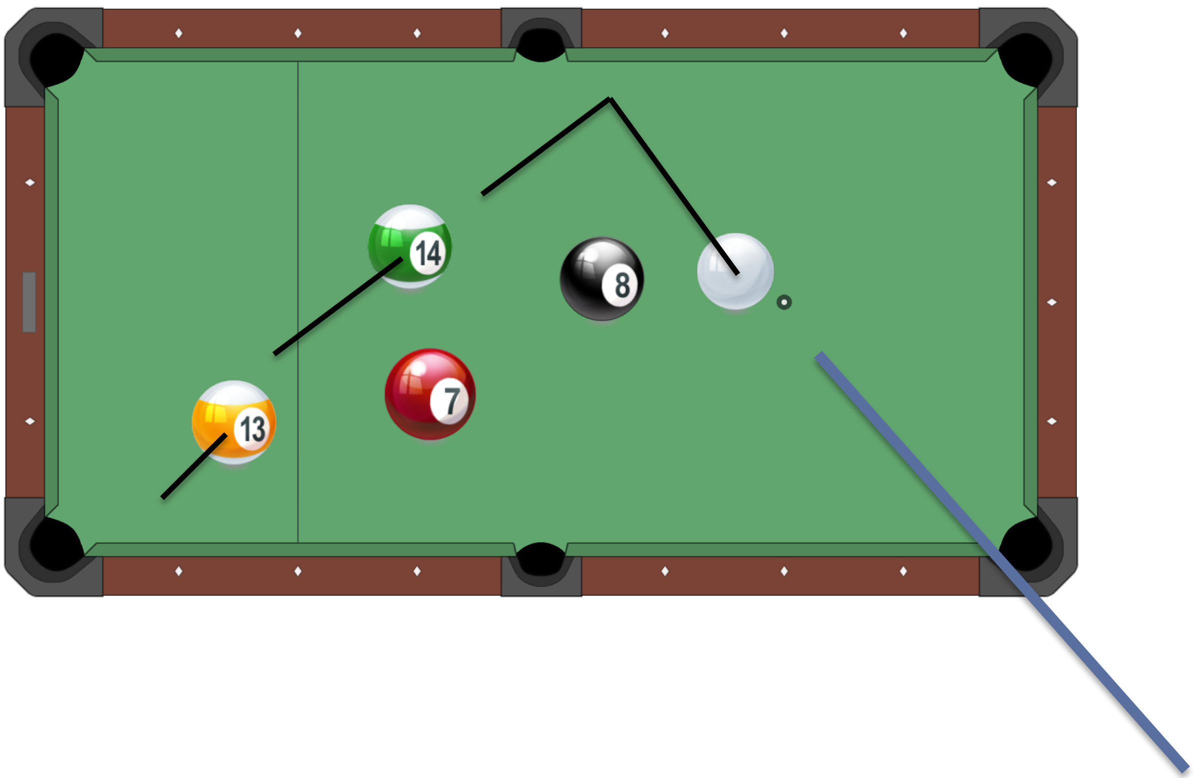
Slide from (Stoyanov & Eisner, 2012)

Error Back-Propagation



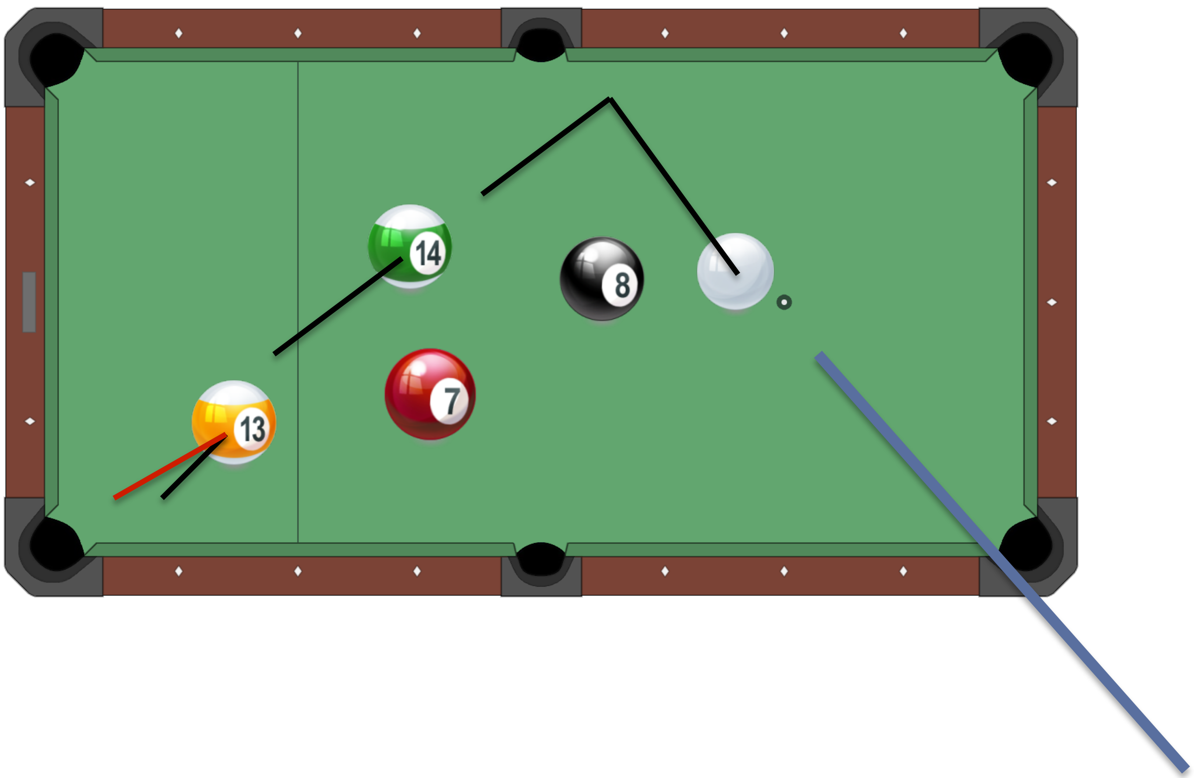
Slide from (Stoyanov & Eisner, 2012)

Error Back-Propagation



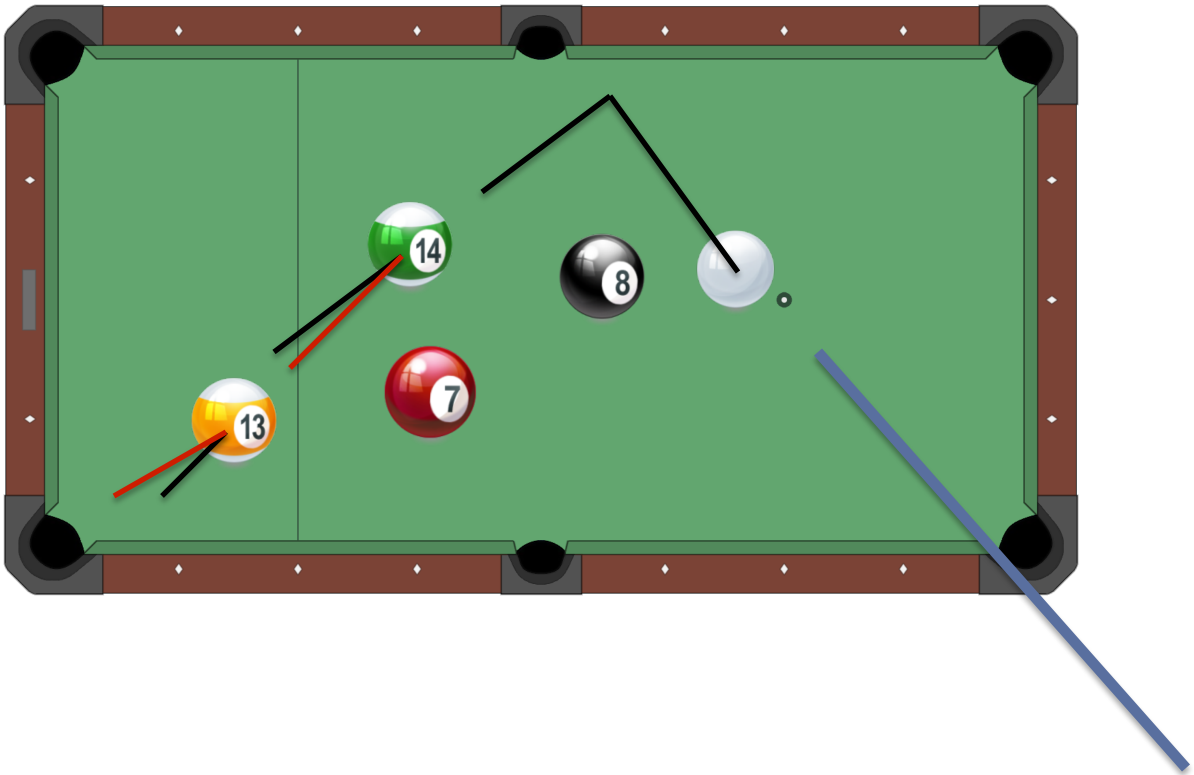
Slide from (Stoyanov & Eisner, 2012)

Error Back-Propagation



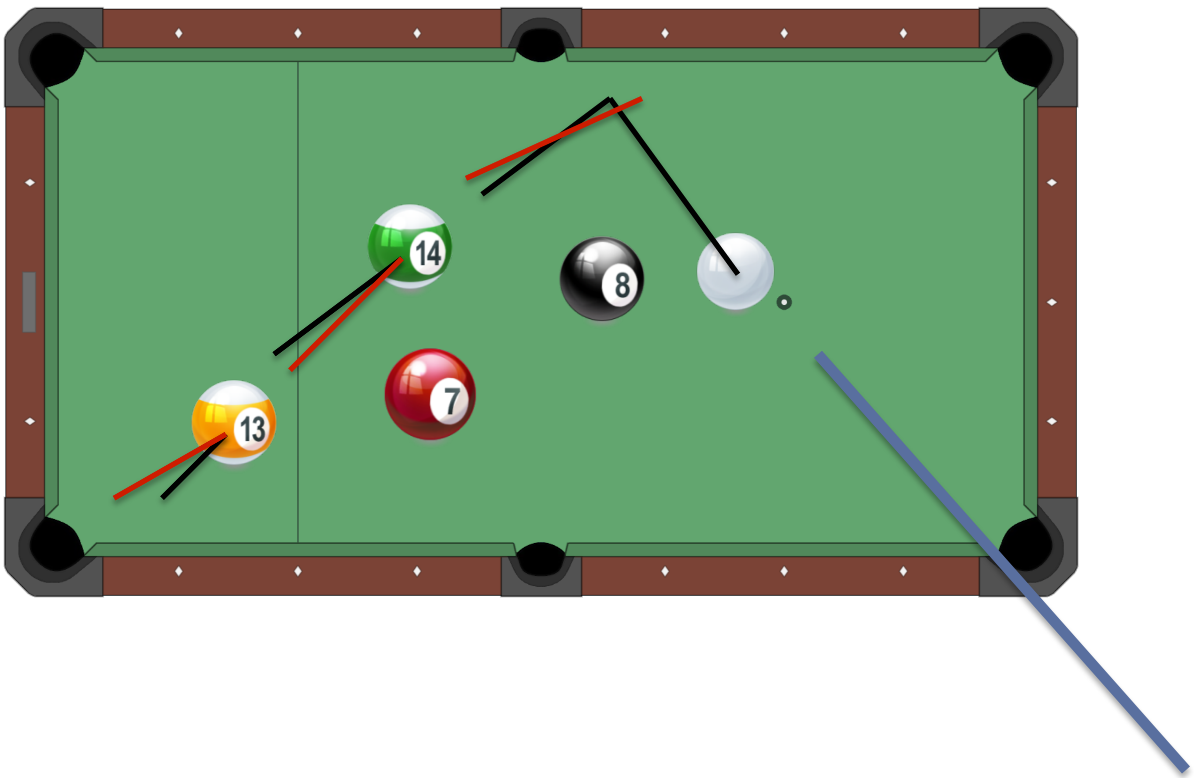
Slide from (Stoyanov & Eisner, 2012)

Error Back-Propagation



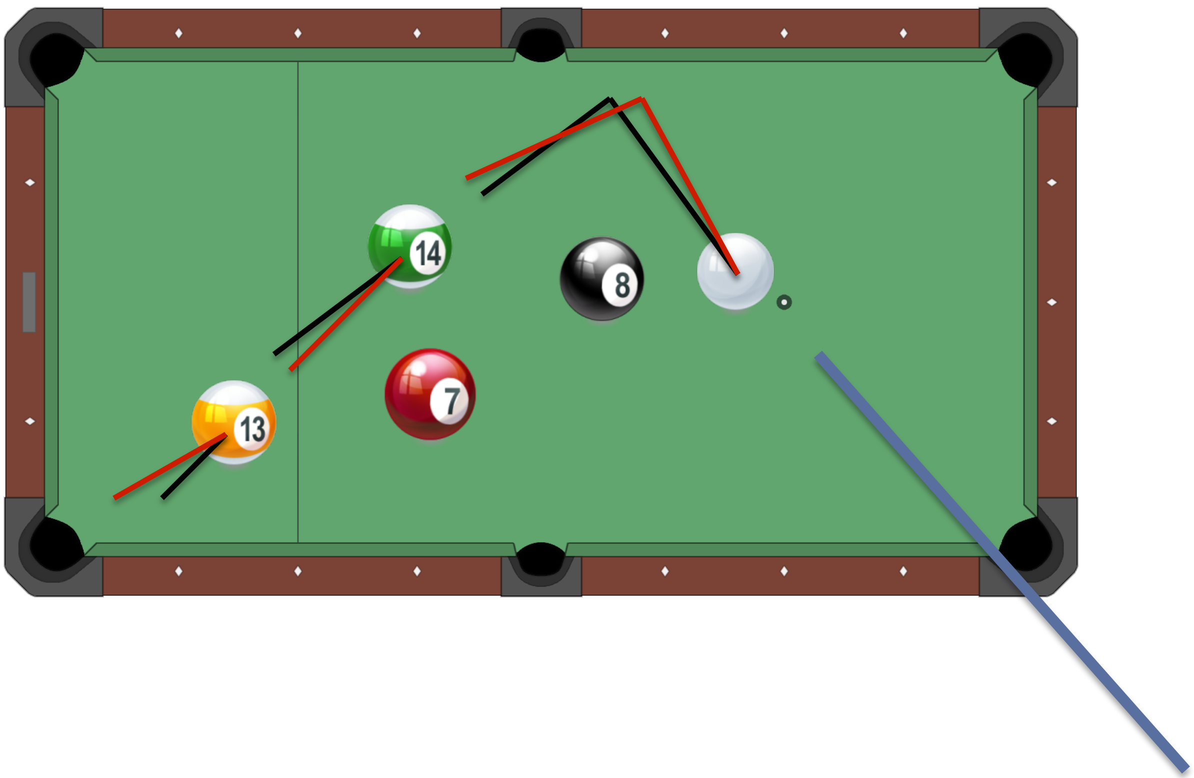
Slide from (Stoyanov & Eisner, 2012)

Error Back-Propagation



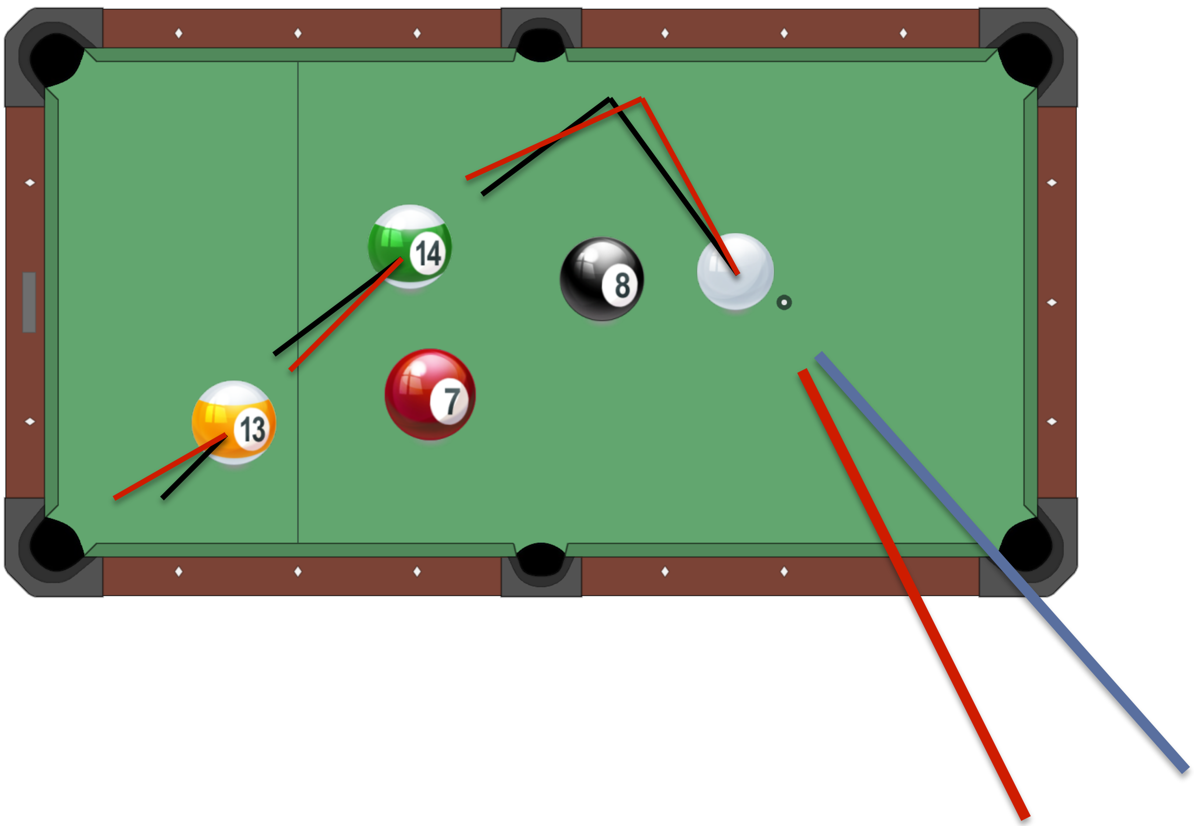
Slide from (Stoyanov & Eisner, 2012)

Error Back-Propagation

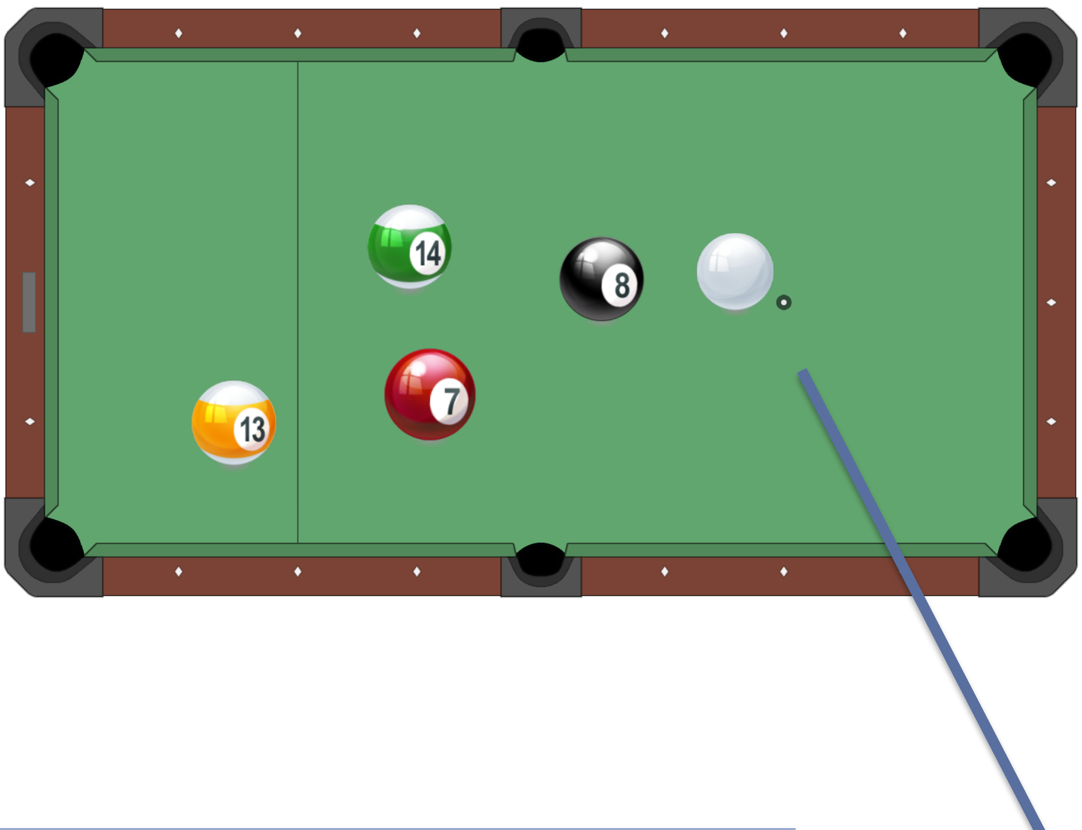


Slide from (Stoyanov & Eisner, 2012)

Error Back-Propagation

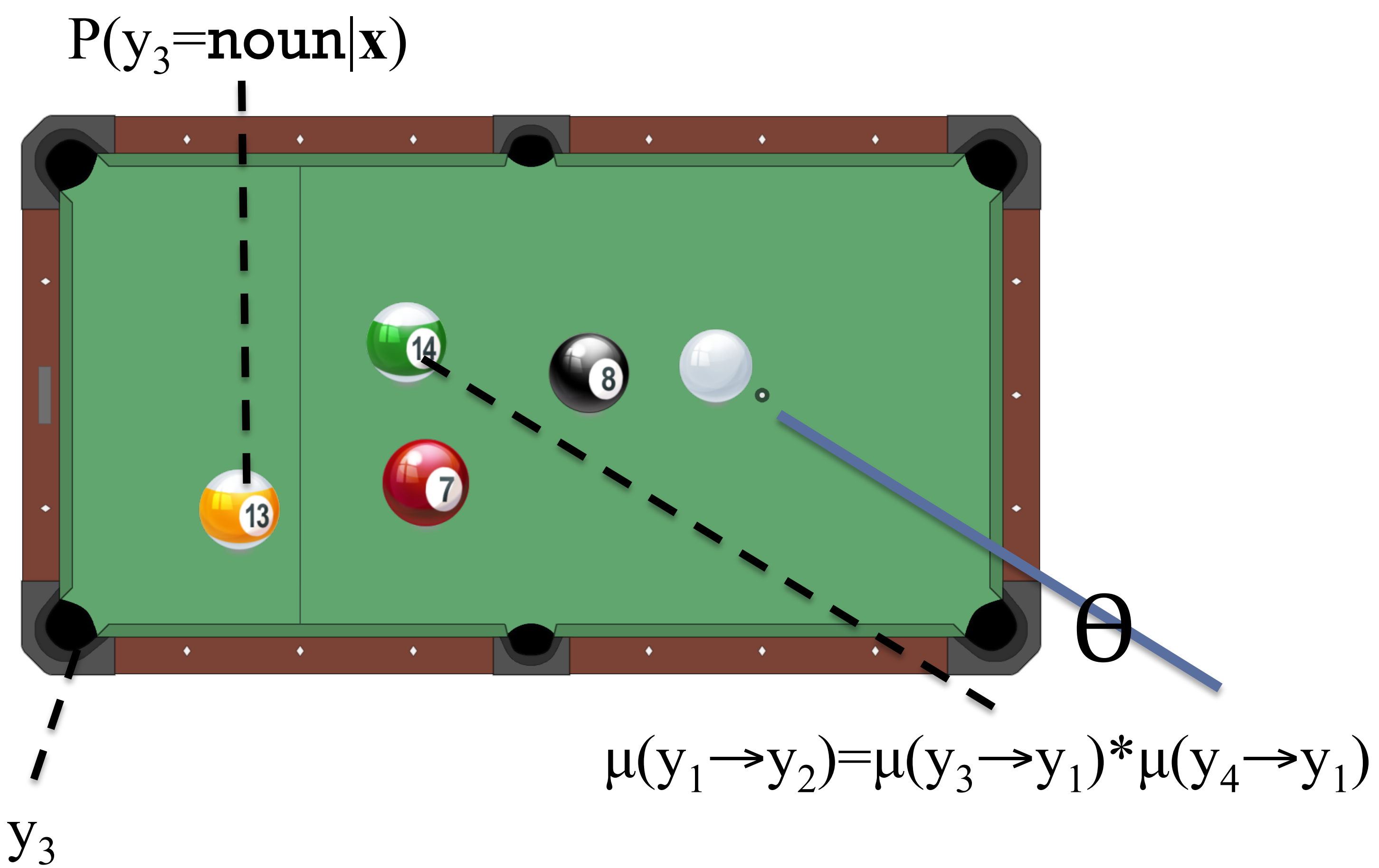


Error Back-Propagation



Slide from (Stoyanov & Eisner, 2012)

Error Back-Propagation



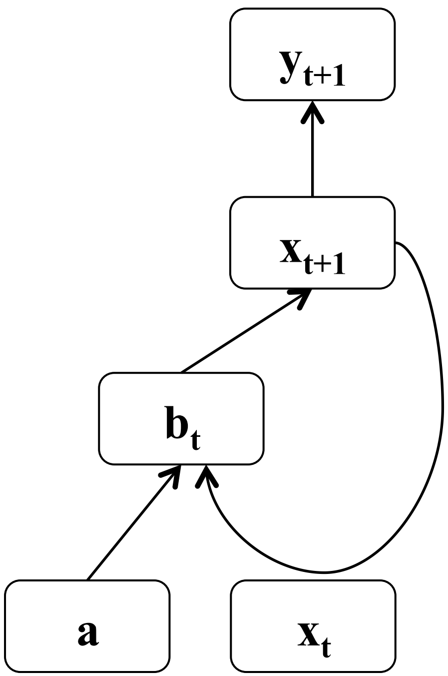
Slide from (Stoyanov & Eisner, 2012)

Error Back-Propagation

- Applying the chain rule of differentiation over and over.
- Forward pass:
  - Regular computation (inference + decoding) in the model (+ remember intermediate quantities).
- Backward pass:
  - Replay the forward pass in reverse, computing gradients.

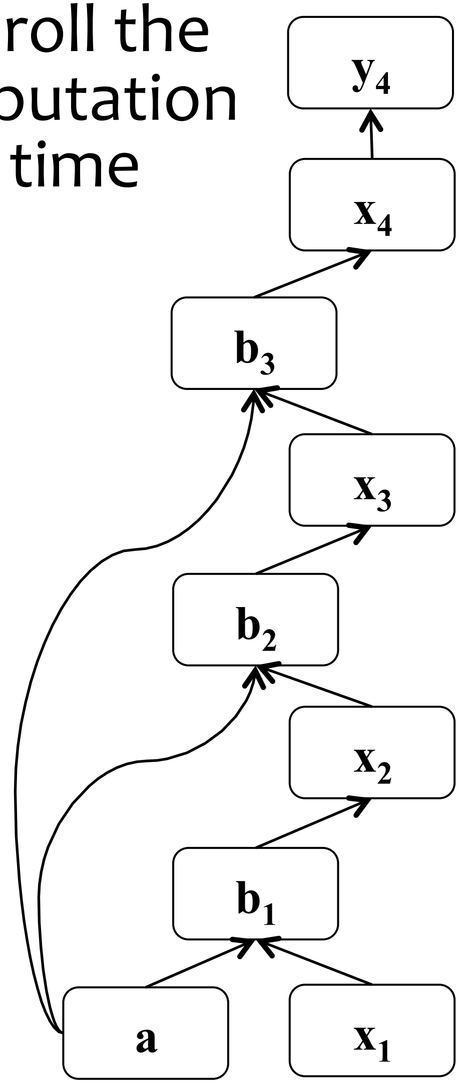
Background: Backprop through time

Recurrent neural network:



BPTT:

1. Unroll the computation over time



2. Run backprop through the resulting feed-forward network

(Robinson & Fallside, 1987)
(Werbos, 1988)
(Mozer, 1995)

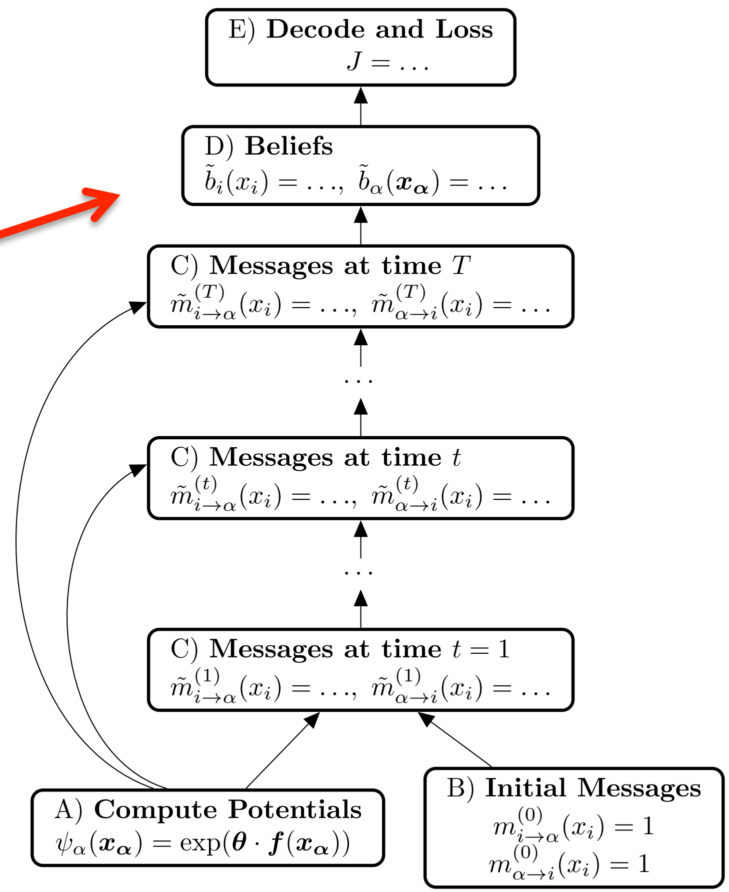
Machine Learning

What went wrong?

How did we compute these **approximate** marginal probabilities anyway?

$$b_\theta(\boldsymbol{y}'_\alpha)$$

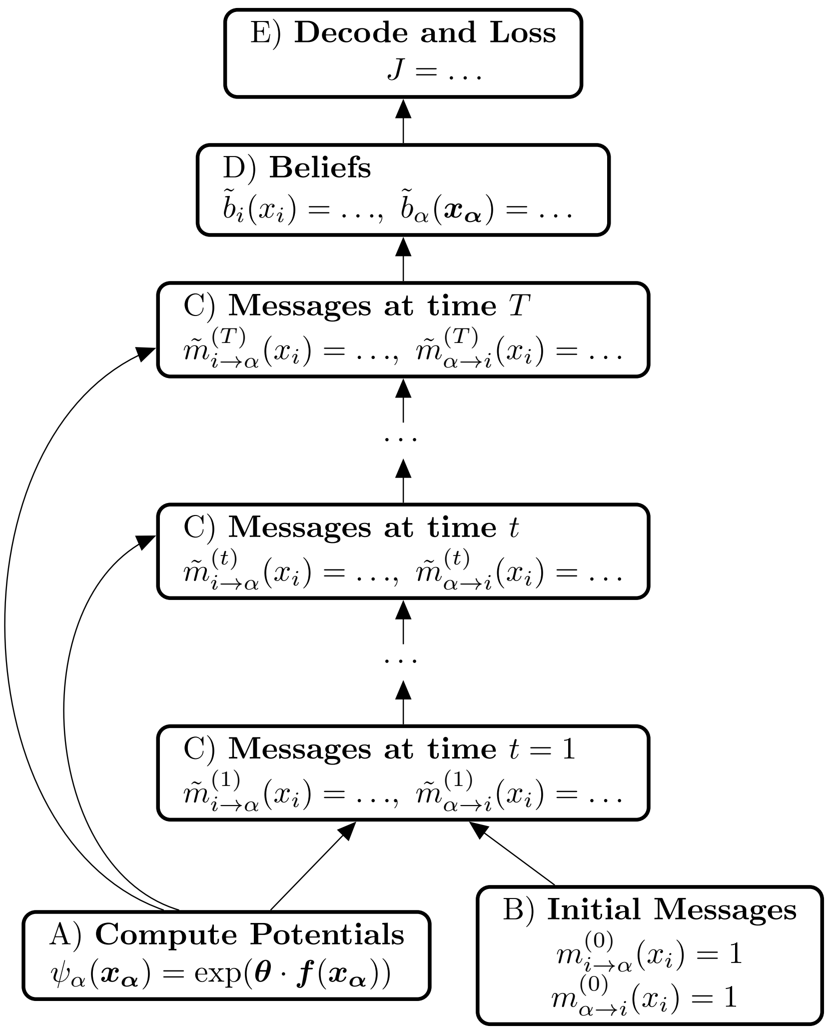
By Belief Propagation of course!



ERMA

Empirical Risk Minimization under Approximations (ERMA)

- Apply *Backprop through time* to Loopy BP
- Unrolls the BP computation graph
- Includes inference, decoding, loss and all the approximations along the way



(Stoyanov, Ropson, & Eisner, 2011)

Machine Learning

JOHNS HOPKINS UNIVERSITY

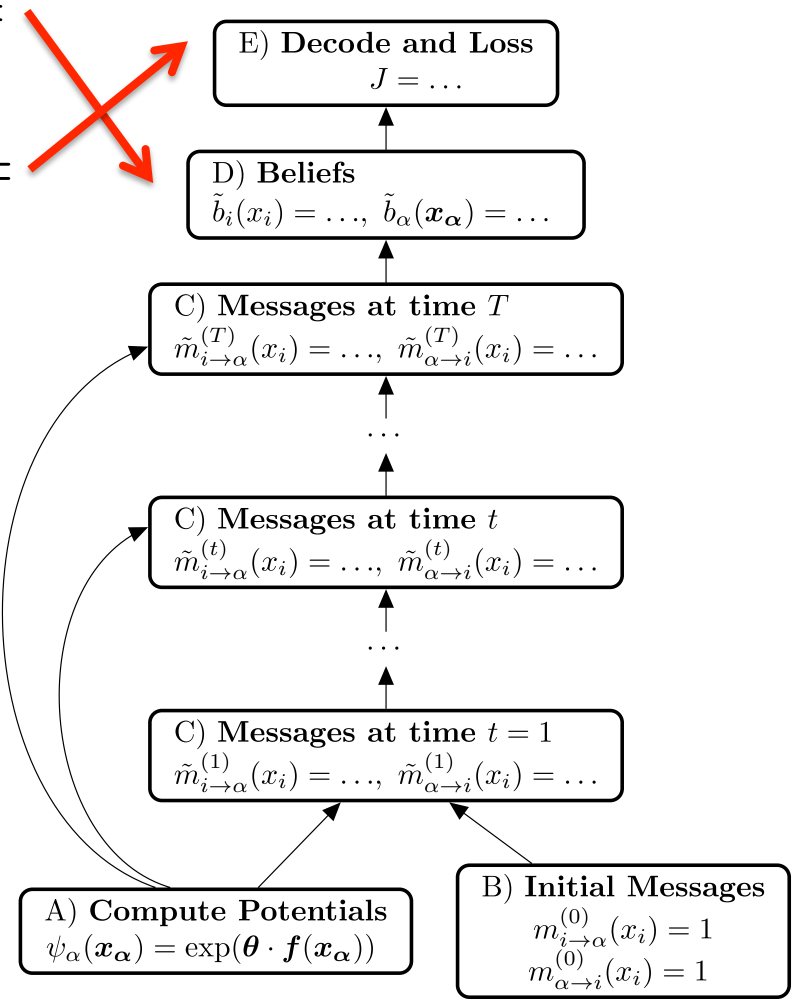
ERMA

Key idea: Open up the black box!

1. Choose **model** to be the computation with all its approximations
2. Choose **objective** to likewise include the approximations
3. Compute **derivative** by backpropagation (treating the entire computation as if it were a neural network)
4. Make no approximations! (Our gradient is exact)

$p_\theta(\boldsymbol{y}) =$

$L(\theta) =$



(Stoyanov, Ropson, & Eisner, 2011)

Machine Learning

JOHNS HOPKINS UNIVERSITY

ERMA

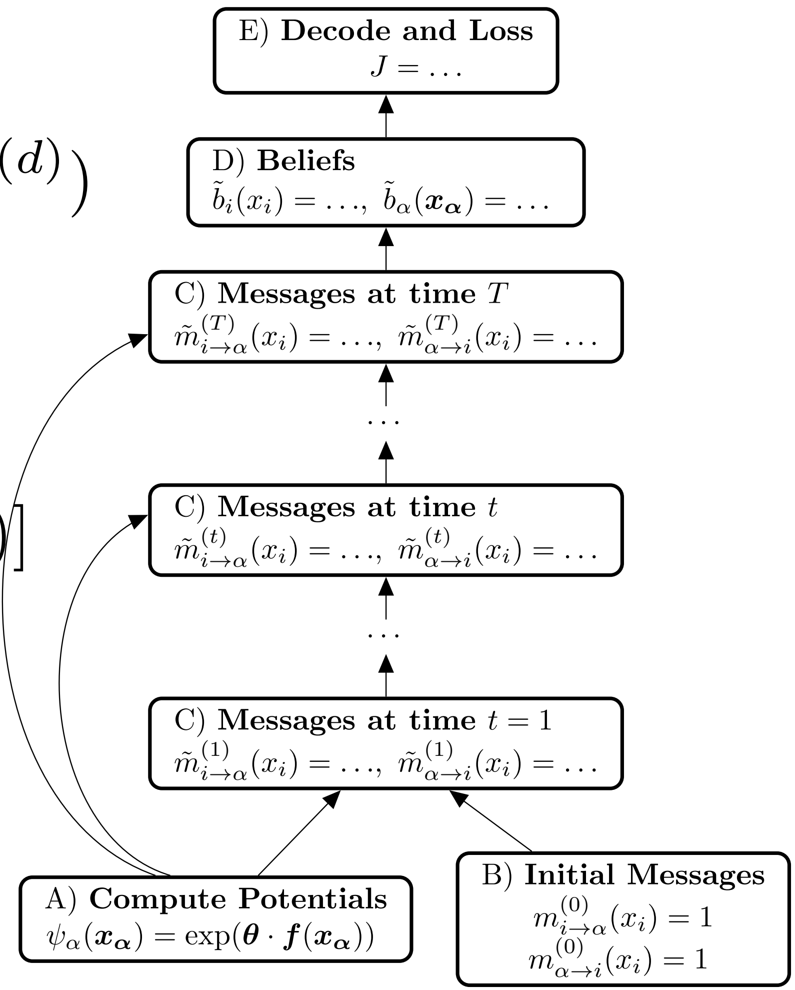
Key idea: Open up the black box!

Empirical Risk Minimization

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \frac{1}{D} \sum_{d=1}^{D} \ell(h_\theta(\boldsymbol{x}^{(d)}), \boldsymbol{y}^{(d)})$$

Minimum Bayes Risk (MBR) Decoder

$$h_\theta(\boldsymbol{x}) = \underset{\boldsymbol{y}}{\operatorname{argmin}} \mathbb{E}_{p_\theta(\boldsymbol{y}'|\boldsymbol{x})}[\ell(\boldsymbol{y}, \boldsymbol{y}')]$$



(Stoyanov, Ropson, & Eisner, 2011)

Machine Learning

Approximation-aware Learning

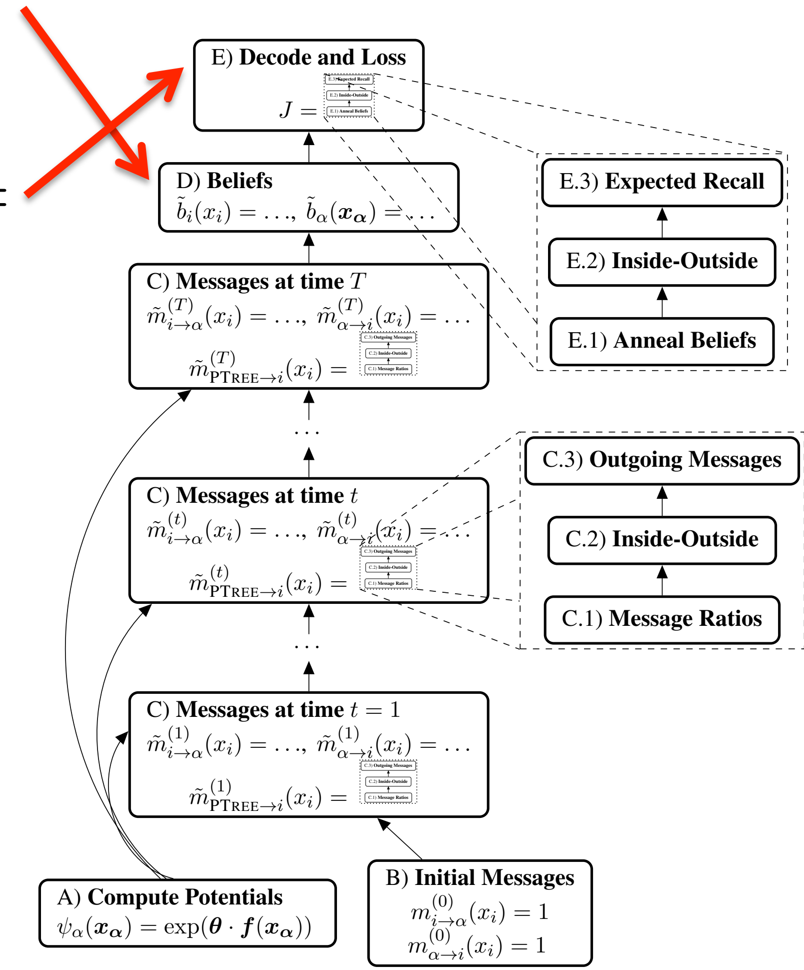
What if we're using Structured BP instead of regular BP?

- No problem, the same approach still applies!
- The only difference is that we **embed dynamic programming algorithms** inside our computation graph.

Key idea: Open up the black box!

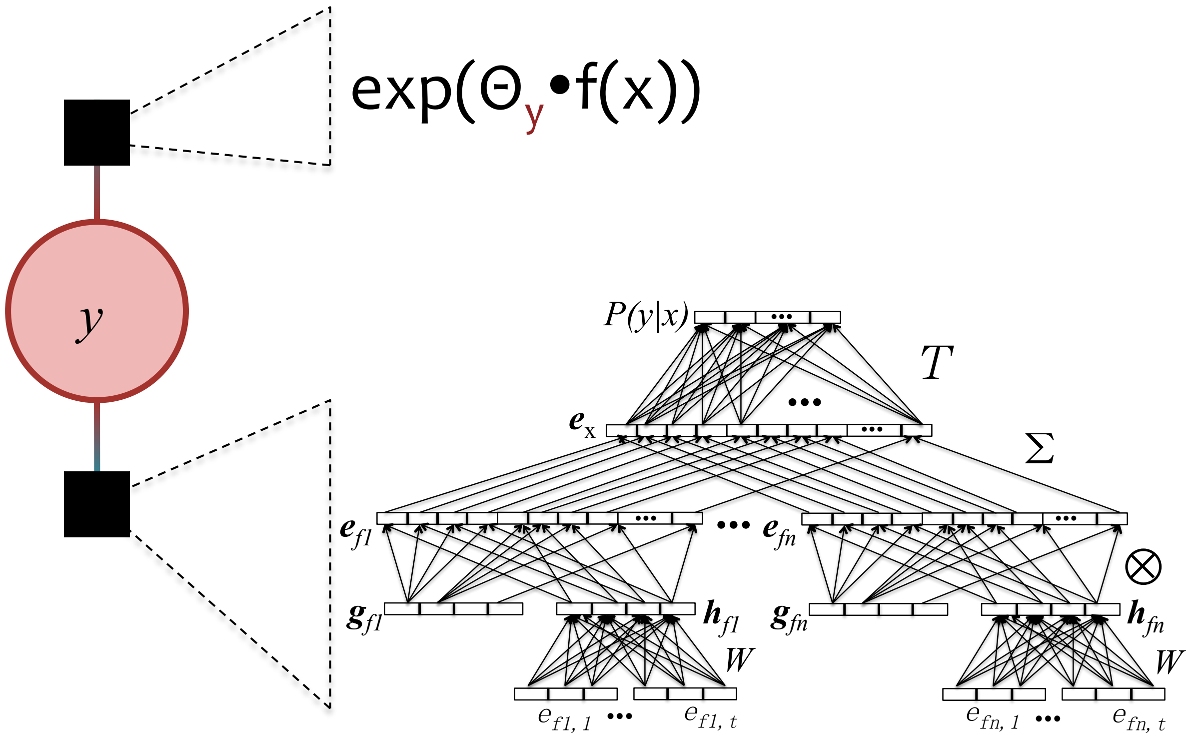
$$p_\theta(\boldsymbol{y}) =$$

$$L(\theta) =$$



(Gormley, Dredze, & Eisner, 2015)

Connection to Deep Learning



(Gormley, Yu, & Dredze, *In submission*)

Empirical Risk Minimization under Approximations (ERMA)

		Approximation Aware	
		No	**Yes**
Loss Aware	**No**	MLE	
	Yes	SVM^{struct} [Finley and Joachims, 2008] M^{3}N [Taskar et al., 2003] Softmax-margin [Gimpel & Smith, 2010]	**ERMA**

Figure from (Stoyanov & Eisner, 2012)

Section 7: Software

Outline

- Do you want to push past the simple NLP models (logistic regression, PCFG, etc.) that we've all been using for 20 years?
- Then this tutorial is extremely practical for you!
  1. **Models:** Factor graphs can express interactions among linguistic structures.
  2. **Algorithm:** BP estimates the global effect of these interactions on each variable, using local computations.
  3. **Intuitions:** What's going on here? Can we trust BP's estimates?
  4. **Fancier Models:** Hide a whole grammar and dynamic programming algorithm within a single factor. BP coordinates multiple factors.
  5. **Tweaked Algorithm:** Finish in fewer steps and make the steps faster.
  6. **Learning:** Tune the parameters. Approximately improve the true predictions -- or truly improve the approximate predictions.
  7. **Software:** Build the model you want!

Outline

- Do you want to push past the simple NLP models (logistic regression, PCFG, etc.) that we've all been using for 20 years?
- Then this tutorial is extremely practical for you!

1. **Models:** Factor graphs can express interactions among linguistic structures.
2. **Algorithm:** BP estimates the global effect of these interactions on each variable, using local computations.
3. **Intuitions:** What's going on here? Can we trust BP's estimates?
4. **Fancier Models:** Hide a whole grammar and dynamic programming algorithm within a single factor. BP coordinates multiple factors.
5. **Tweaked Algorithm:** Finish in fewer steps and make the steps faster.
6. **Learning:** Tune the parameters. Approximately improve the true predictions -- or truly improve the approximate predictions.
7. **Software:** Build the model you want!

Pacaya

Features:

- Structured Loopy BP over factor graphs with:
  - Discrete variables
  - Structured constraint factors (e.g. projective dependency tree constraint factor)
  - ERMA training with backpropagation
  - Backprop through structured factors (Gormley, Dredze, & Eisner, 2015)

Language: Java

Authors: Gormley, Mitchell, & Wolfe

URL: http://www.cs.jhu.edu/~mrg/software/

(Gormley, Mitchell, Van Durme, & Dredze, 2014)
(Gormley, Dredze, & Eisner, 2015)

ERMA

Features:
ERMA performs inference and training on CRFs and MRFs with arbitrary model structure over discrete variables. The training regime, Empirical Risk Minimization under Approximations is loss-aware and approximation-aware. ERMA can optimize several loss functions such as Accuracy, MSE and F-score.

Language: Java

Authors: Stoyanov

URL: https://sites.google.com/site/ermasoftware/

(Stoyanov, Ropson, & Eisner, 2011)
(Stoyanov & Eisner, 2012)

Graphical Models Libraries

- **Factorie** (McCallum, Shultz, & Singh, 2012) is a Scala library allowing modular specification of inference, learning, and optimization methods. Inference algorithms include belief propagation and MCMC. Learning settings include maximum likelihood learning, maximum margin learning, learning with approximate inference, SampleRank, pseudo-likelihood.
http://factorie.cs.umass.edu/

- **LibDAI** (Mooij, 2010) is a C++ library that supports inference, but not learning, via Loopy BP, Fractional BP, Tree-Reweighted BP, (Double-loop) Generalized BP, variants of Loop Corrected Belief Propagation, Conditioned Belief Propagation, and Tree Expectation Propagation.
http://www.libdai.org

- **OpenGM2** (Andres, Beier, & Kappes, 2012) provides a C++ template library for discrete factor graphs with support for learning and inference (including tie-ins to all LibDAI inference algorithms).
http://hci.iwr.uni-heidelberg.de/opengm2/

- **FastInf** (Jaimovich, Meshi, Mcgraw, Elidan) is an efficient Approximate Inference Library in C++.
http://compbio.cs.huji.ac.il/FastInf/fastInf/FastInf_Homepage.html

- **Infer.NET** (Minka et al., 2012) is a .NET language framework for graphical models with support for Expectation Propagation and Variational Message Passing.
http://research.microsoft.com/en-us/um/cambridge/projects/infernet

References

- M. Auli and A. Lopez, “A Comparison of Loopy Belief Propagation and Dual Decomposition for Integrated CCG Supertagging and Parsing,” in Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Portland, Oregon, USA, 2011, pp. 470–480.
- M. Auli and A. Lopez, “Training a Log-Linear Parser with Loss Functions via Softmax-Margin,” in Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing, Edinburgh, Scotland, UK., 2011, pp. 333–343.
- Y. Bengio, “Training a neural network with a financial criterion rather than a prediction criterion,” in Decision Technologies for Financial Engineering: Proceedings of the Fourth International Conference on Neural Networks in the Capital Markets (NNCM’96), World Scientific Publishing, 1997, pp. 36–48.
- D. P. Bertsekas and J. N. Tsitsiklis, Parallel and distributed computation: numerical methods. Prentice-Hall, Inc., 1989.
- D. P. Bertsekas and J. N. Tsitsiklis, Parallel and distributed computation: numerical methods. Athena Scientific, 1997.
- L. Bottou and P. Gallinari, “A Framework for the Cooperation of Learning Algorithms,” in Advances in Neural Information Processing Systems, vol. 3, D. Touretzky and R. Lippmann, Eds. Denver: Morgan Kaufmann, 1991.
- R. Bunescu and R. J. Mooney, “Collective information extraction with relational Markov networks,” 2004, p. 438–es.
- C. Burfoot, S. Bird, and T. Baldwin, “Collective Classification of Congressional Floor-Debate Transcripts,” presented at the Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Techologies, 2011, pp. 1506–1515.
- D. Burkett and D. Klein, “Fast Inference in Phrase Extraction Models with Belief Propagation,” presented at the Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, 2012, pp. 29–38.
- T. Cohn and P. Blunsom, “Semantic Role Labelling with Tree Conditional Random Fields,” presented at the Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005), 2005, pp. 169–172.

- F. Cromierès and S. Kurohashi, “An Alignment Algorithm Using Belief Propagation and a Structure-Based Distortion Model,” in Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009), Athens, Greece, 2009, pp. 166–174.
- M. Dreyer, “A non-parametric model for the discovery of inflectional paradigms from plain text using graphical models over strings,” Johns Hopkins University, Baltimore, MD, USA, 2011.
- M. Dreyer and J. Eisner, “Graphical Models over Multiple Strings,” presented at the Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing, 2009, pp. 101–110.
- M. Dreyer and J. Eisner, “Discovering Morphological Paradigms from Plain Text Using a Dirichlet Process Mixture Model,” presented at the Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing, 2011, pp. 616–627.
- J. Duchi, D. Tarlow, G. Elidan, and D. Koller, “Using Combinatorial Optimization within Max-Product Belief Propagation,” Advances in neural information processing systems, 2006.
- G. Durrett, D. Hall, and D. Klein, “Decentralized Entity-Level Modeling for Coreference Resolution,” presented at the Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), 2013, pp. 114–124.
- G. Elidan, I. McGraw, and D. Koller, “Residual belief propagation: Informed scheduling for asynchronous message passing,” in Proceedings of the Twenty-second Conference on Uncertainty in AI (UAI, 2006.
- K. Gimpel and N. A. Smith, “Softmax-Margin CRFs: Training Log-Linear Models with Cost Functions,” in Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Los Angeles, California, 2010, pp. 733–736.
- J. Gonzalez, Y. Low, and C. Guestrin, “Residual splash for optimally parallelizing belief propagation,” in International Conference on Artificial Intelligence and Statistics, 2009, pp. 177–184.
- D. Hall and D. Klein, “Training Factored PCFGs with Expectation Propagation,” presented at the Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, 2012, pp. 1146–1156.

JOHNS HOPKINS UNIVERSITY

- T. Heskes, “Stable fixed points of loopy belief propagation are minima of the Bethe free energy,” Advances in Neural Information Processing Systems, vol. 15, pp. 359–366, 2003.
- T. Heskes and O. Zoeter, “Expectation propagation for approximate inference in dynamic Bayesian networks,” Uncertainty in Artificial Intelligence, 2002, pp. 216-233.
- A. T. Ihler, J. W. Fisher III, A. S. Willsky, and D. M. Chickering, “Loopy belief propagation: convergence and effects of message errors.,” Journal of Machine Learning Research, vol. 6, no. 5, 2005.
- A. T. Ihler and D. A. McAllester, “Particle belief propagation,” in International Conference on Artificial Intelligence and Statistics, 2009, pp. 256–263.
- J. Jancsary, J. Matiasek, and H. Trost, “Revealing the Structure of Medical Dictations with Conditional Random Fields,” presented at the Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing, 2008, pp. 1–10.
- J. Jiang, T. Moon, H. Daumé III, and J. Eisner, “Prioritized Asynchronous Belief Propagation,” in ICML Workshop on Inferning, 2013.
- A. Kazantseva and S. Szpakowicz, “Linear Text Segmentation Using Affinity Propagation,” presented at the Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing, 2011, pp. 284–293.
- T. Koo and M. Collins, “Hidden-Variable Models for Discriminative Reranking,” presented at the Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing, 2005, pp. 507–514.
- A. Kulesza and F. Pereira, “Structured Learning with Approximate Inference.,” in NIPS, 2007, vol. 20, pp. 785–792.
- J. Lee, J. Naradowsky, and D. A. Smith, “A Discriminative Model for Joint Morphological Disambiguation and Dependency Parsing,” in Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Portland, Oregon, USA, 2011, pp. 885–894.
- S. Lee, “Structured Discriminative Model For Dialog State Tracking,” presented at the Proceedings of the SIGDIAL 2013 Conference, 2013, pp. 442–451.

- X. Liu, M. Zhou, X. Zhou, Z. Fu, and F. Wei, “Joint Inference of Named Entity Recognition and Normalization for Tweets,” presented at the Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), 2012, pp. 526–535.
- D. J. C. MacKay, J. S. Yedidia, W. T. Freeman, and Y. Weiss, “A Conversation about the Bethe Free Energy and Sum-Product,” MERL, TR2001-18, 2001.
- A. Martins, N. Smith, E. Xing, P. Aguiar, and M. Figueiredo, “Turbo Parsers: Dependency Parsing by Approximate Variational Inference,” presented at the Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing, 2010, pp. 34–44.
- D. McAllester, M. Collins, and F. Pereira, “Case-Factor Diagrams for Structured Probabilistic Modeling,” in In Proceedings of the Twentieth Conference on Uncertainty in Artificial Intelligence (UAI’04), 2004.
- T. Minka, “Divergence measures and message passing,” Technical report, Microsoft Research, 2005.
- T. P. Minka, “Expectation propagation for approximate Bayesian inference,” in Uncertainty in Artificial Intelligence, 2001, vol. 17, pp. 362–369.
- M. Mitchell, J. Aguilar, T. Wilson, and B. Van Durme, “Open Domain Targeted Sentiment,” presented at the Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, 2013, pp. 1643–1654.
- K. P. Murphy, Y. Weiss, and M. I. Jordan, “Loopy belief propagation for approximate inference: An empirical study,” in *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, 1999, pp. 467–475.
- T. Nakagawa, K. Inui, and S. Kurohashi, “Dependency Tree-based Sentiment Classification using CRFs with Hidden Variables,” presented at the Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics, 2010, pp. 786–794.
- J. Naradowsky, S. Riedel, and D. Smith, “Improving NLP through Marginalization of Hidden Syntactic Structure,” in Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, 2012, pp. 810–820.
- J. Naradowsky, T. Vieira, and D. A. Smith, Grammarless Parsing for Joint Inference. Mumbai, India, 2012.
- J. Niehues and S. Vogel, “Discriminative Word Alignment via Alignment Matrix Modeling,” presented at the Proceedings of the Third Workshop on Statistical Machine Translation, 2008, pp. 18–25.

- J. Pearl, Probabilistic reasoning in intelligent systems: networks of plausible inference. Morgan Kaufmann, 1988.
- X. Pitkow, Y. Ahmadian, and K. D. Miller, “Learning unbelievable probabilities,” in Advances in Neural Information Processing Systems 24, J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2011, pp. 738–746.
- V. Qazvinian and D. R. Radev, “Identifying Non-Explicit Citing Sentences for Citation-Based Summarization.,” presented at the Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, 2010, pp. 555–564.
- H. Ren, W. Xu, Y. Zhang, and Y. Yan, “Dialog State Tracking using Conditional Random Fields,” presented at the Proceedings of the SIGDIAL 2013 Conference, 2013, pp. 457–461.
- D. Roth and W. Yih, “Probabilistic Reasoning for Entity & Relation Recognition,” presented at the COLING 2002: The 19th International Conference on Computational Linguistics, 2002.
- A. Rudnick, C. Liu, and M. Gasser, “HLTDI: CL-WSD Using Markov Random Fields for SemEval-2013 Task 10,” presented at the Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013), 2013, pp. 171–177.
- T. Sato, “Inside-Outside Probability Computation for Belief Propagation.,” in IJCAI, 2007, pp. 2605–2610.
- D. A. Smith and J. Eisner, “Dependency Parsing by Belief Propagation,” in Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), Honolulu, 2008, pp. 145–156.
- V. Stoyanov and J. Eisner, “Fast and Accurate Prediction via Evidence-Specific MRF Structure,” in ICML Workshop on Inferning: Interactions between Inference and Learning, Edinburgh, 2012.
- V. Stoyanov and J. Eisner, “Minimum-Risk Training of Approximate CRF-Based NLP Systems,” in Proceedings of NAACL-HLT, 2012, pp. 120–130.

- V. Stoyanov, A. Ropson, and J. Eisner, "Empirical Risk Minimization of Graphical Model Parameters Given Approximate Inference, Decoding, and Model Structure," in Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS), Fort Lauderdale, 2011, vol. 15, pp. 725–733.
- E. B. Sudderth, A. T. Ihler, W. T. Freeman, and A. S. Willsky, "Nonparametric belief propagation," MIT, Technical Report 2551, 2002.
- E. B. Sudderth, A. T. Ihler, W. T. Freeman, and A. S. Willsky, "Nonparametric belief propagation," in In Proceedings of CVPR, 2003.
- E. B. Sudderth, A. T. Ihler, M. Isard, W. T. Freeman, and A. S. Willsky, "Nonparametric belief propagation," Communications of the ACM, vol. 53, no. 10, pp. 95–103, 2010.
- C. Sutton and A. McCallum, "Collective Segmentation and Labeling of Distant Entities in Information Extraction," in ICML Workshop on Statistical Relational Learning and Its Connections to Other Fields, 2004.
- C. Sutton and A. McCallum, "Piecewise Training of Undirected Models," in Conference on Uncertainty in Artificial Intelligence (UAI), 2005.
- C. Sutton and A. McCallum, "Improved dynamic schedules for belief propagation," UAI, 2007.
- M. J. Wainwright, T. Jaakkola, and A. S. Willsky, "Tree-based reparameterization for approximate inference on loopy graphs.," in NIPS, 2001, pp. 1001–1008.
- Z. Wang, S. Li, F. Kong, and G. Zhou, "Collective Personal Profile Summarization with Social Networks," presented at the Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, 2013, pp. 715–725.
- Y. Watanabe, M. Asahara, and Y. Matsumoto, "A Graph-Based Approach to Named Entity Categorization in Wikipedia Using Conditional Random Fields," presented at the Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL), 2007, pp. 649–657.

- Y. Weiss and W. T. Freeman, “On the optimality of solutions of the max-product belief-propagation algorithm in arbitrary graphs,” Information Theory, IEEE Transactions on, vol. 47, no. 2, pp. 736–744, 2001.
- J. S. Yedidia, W. T. Freeman, and Y. Weiss, “Bethe free energy, Kikuchi approximations, and belief propagation algorithms,” MERL, TR2001-16, 2001.
- J. S. Yedidia, W. T. Freeman, and Y. Weiss, “Constructing free-energy approximations and generalized belief propagation algorithms,” IEEE Transactions on Information Theory, vol. 51, no. 7, pp. 2282–2312, Jul. 2005.
- J. S. Yedidia, W. T. Freeman, and Y. Weiss, “Generalized belief propagation,” in NIPS, 2000, vol. 13, pp. 689–695.
- J. S. Yedidia, W. T. Freeman, and Y. Weiss, “Understanding belief propagation and its generalizations,” Exploring artificial intelligence in the new millennium, vol. 8, pp. 236–239, 2003.
- J. S. Yedidia, W. T. Freeman, and Y. Weiss, “Constructing Free Energy Approximations and Generalized Belief Propagation Algorithms,” MERL, TR-2004-040, 2004.
- J. S. Yedidia, W. T. Freeman, and Y. Weiss, “Constructing free-energy approximations and generalized belief propagation algorithms,” Information Theory, IEEE Transactions on, vol. 51, no. 7, pp. 2282–2312, 2005.