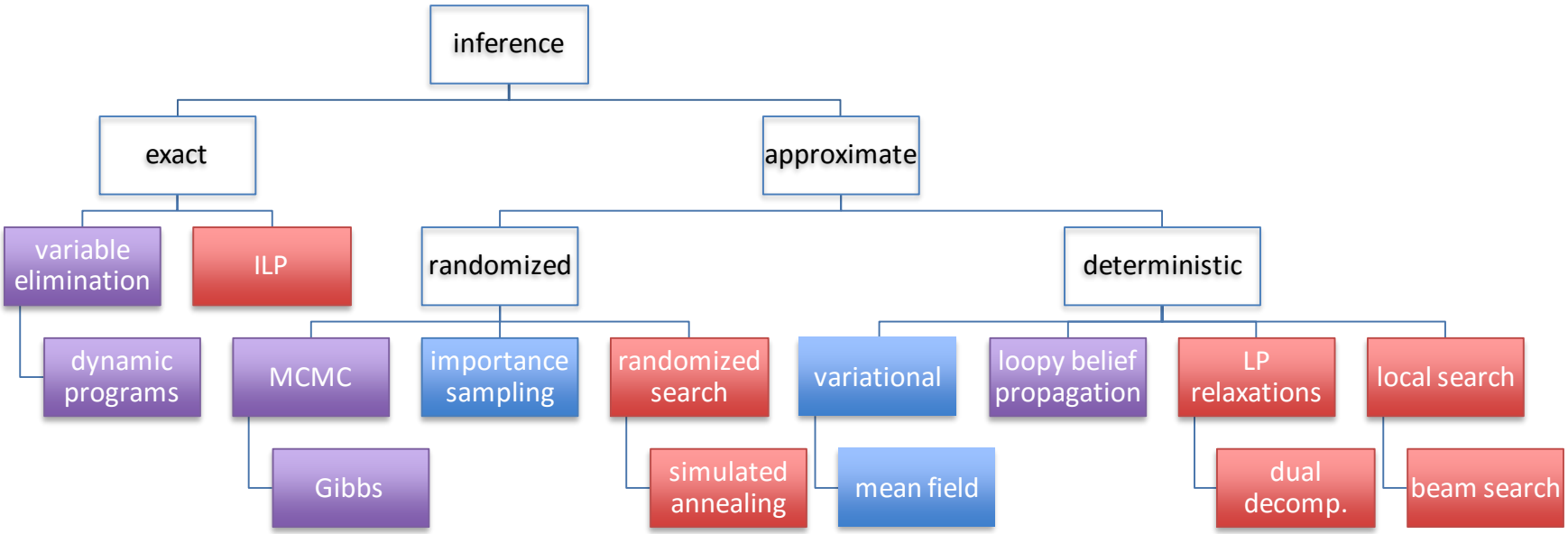
Approaches to Inference



red = hard inference
blue = soft inference
purple = both

Hidden Markov Model

- **X** and **Y** are both sequences of symbols
  - **X** is a sequence from the vocabulary Σ
  - **Y** is a sequence from the state space Λ

$$p(\boldsymbol{Y} = \boldsymbol{s}, \boldsymbol{X} = \boldsymbol{w}) =$$
$$p(\text{start}, s_1, w_1, s_2, w_2, \ldots, s_n, w_n \text{stop}) \quad = \quad \prod_{i=1}^{n+1} \eta(w_i \mid s_i) \times \gamma(s_i \mid s_{i-1})$$

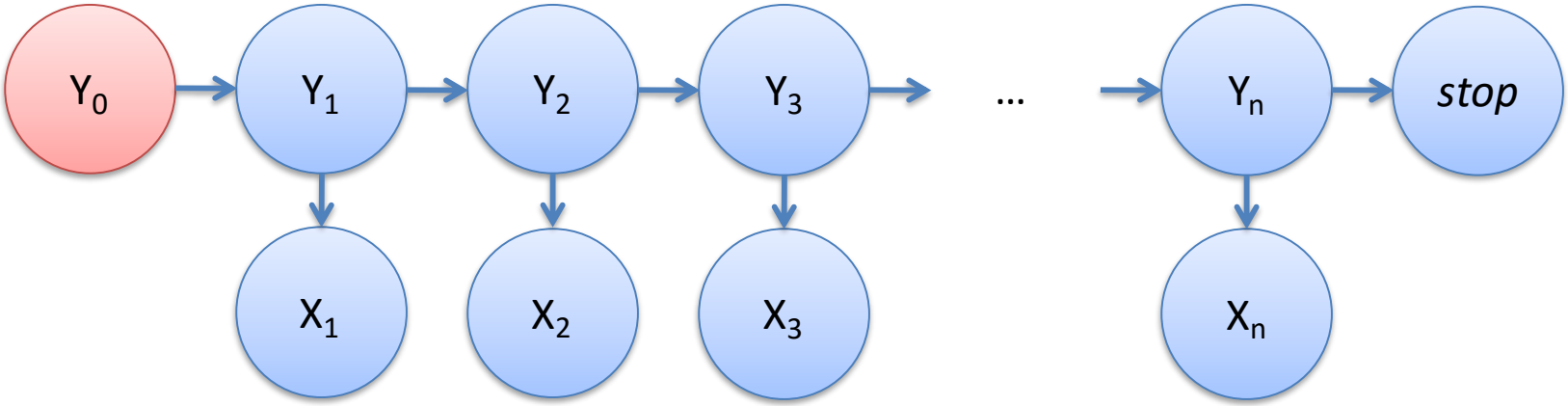
- Parameters:
  - Transitions **γ** including γ(*stop* | s), γ(s | *start*)
  - Emissions **η**

Hidden Markov Model

- The joint model's independence assumptions are easy to capture with a Bayesian network.

$$p(\boldsymbol{Y} = \boldsymbol{s}, \boldsymbol{X} = \boldsymbol{w}) =$$

$$p(\text{start}, s_1, w_1, s_2, w_2, \ldots, s_n, w_n \text{stop}) \quad = \quad \prod_{i=1}^{n+1} \eta(w_i \mid s_i) \times \gamma(s_i \mid s_{i-1})$$

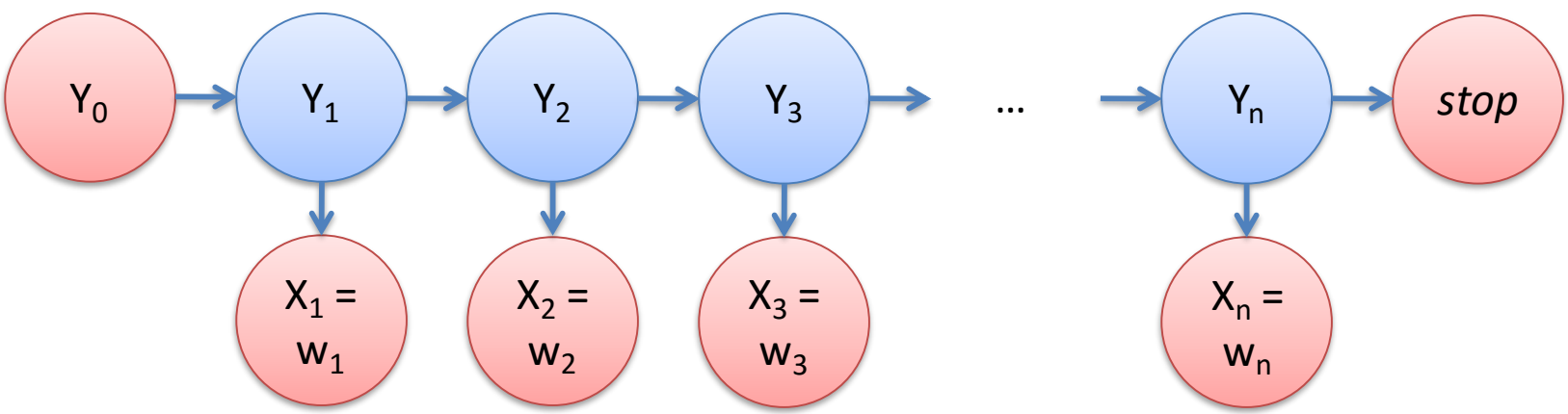


Hidden Markov Model

- The MPE/MAP inference problem is to find the most probable value of **Y** given **X** = **x**.

$$p(\boldsymbol{Y} = \boldsymbol{s}, \boldsymbol{X} = \boldsymbol{w}) =$$

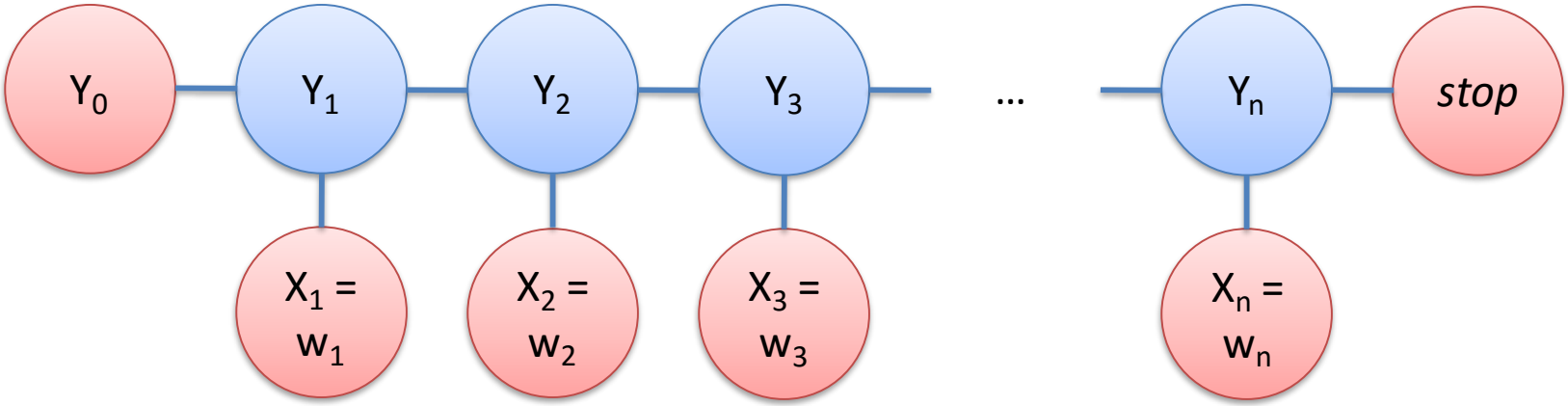
$$p(\text{start}, s_1, w_1, s_2, w_2, \ldots, s_n, w_n \text{stop}) \quad = \quad \prod_{i=1}^{n+1} \eta(w_i \mid s_i) \times \gamma(s_i \mid s_{i-1})$$



Hidden Markov Model

- The MPE/MAP inference problem is to find the most probable value of **Y** given **X** = **x**.

- Markov network:



Markov Network

- A different graphical model representation; undirected. Vertices are still r.v.s.
- Every clique C in the graph gets a *local* scoring function ϕ_C that maps assignments to values.

$$mulscore(\boldsymbol{x}, \boldsymbol{y}) = \prod_{C \in \mathcal{C}} \phi_C(\Pi_C(\boldsymbol{x}, \boldsymbol{y}))$$

$$addscore(\boldsymbol{x}, \boldsymbol{y}) = \sum_{C \in \mathcal{C}} \log \phi_C(\Pi_C(\boldsymbol{x}, \boldsymbol{y}))$$

- This score can be *globally* renormalized to obtain a probabilistic interpretation. (Not today.)

Restriction #1

1. The score function *factors locally*.
   – The more locally, the better!

$$score(\boldsymbol{x}, \boldsymbol{y}) = \sum_{C \in \mathcal{C}} \log \phi_C(\Pi_C(\boldsymbol{x}, \boldsymbol{y}))$$

Linear Models

- Define a feature vector function **g** that maps (**x**, **y**) pairs into d-dimensional real space.
- Score is linear in **g**(**x**, **y**).

$$
\begin{aligned}
score(\boldsymbol{x}, \boldsymbol{y}) &= \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y}) \\
\boldsymbol{y}^* &= \arg\max_{\boldsymbol{y} \in \mathcal{Y}_{\boldsymbol{x}}} \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y})
\end{aligned}
$$

- Results:
  - decoding seeks **y** to maximize the score.
  - learning seeks **w** to … do something we'll talk about later.
- Extremely general!

Generic Noisy Channel as Linear Model

$$
\begin{aligned}
\hat{\boldsymbol{y}} &= \arg\max_{\boldsymbol{y}} \log\left(p(\boldsymbol{y}) \cdot p(\boldsymbol{x} \mid \boldsymbol{y})\right) \\
&= \arg\max_{\boldsymbol{y}} \log p(\boldsymbol{y}) + \log p(\boldsymbol{x} \mid \boldsymbol{y}) \\
&= \arg\max_{\boldsymbol{y}} w_{\boldsymbol{y}} + w_{\boldsymbol{x}|\boldsymbol{y}} \\
&= \arg\max_{\boldsymbol{y}} \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y})
\end{aligned}
$$

- Of course, the two probability terms are typically composed of "smaller" factors; each can be understood as an exponentiated weight.

Max Ent Models as Linear Models

$$
\begin{aligned}
\hat{\boldsymbol{y}} &= \arg\max_{\boldsymbol{y}} \log p(\boldsymbol{y} \mid \boldsymbol{x}) \\
&= \arg\max_{\boldsymbol{y}} \log \frac{\exp \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y})}{z(\boldsymbol{x})} \\
&= \arg\max_{\boldsymbol{y}} \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y}) - \log z(\boldsymbol{x}) \\
&= \arg\max_{\boldsymbol{y}} \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y})
\end{aligned}
$$

HMMs as Linear Models

$$
\begin{aligned}
\hat{\boldsymbol{y}} &= \arg\max_{\boldsymbol{y}} \log p(\boldsymbol{x}, \boldsymbol{y}) \\
&= \arg\max_{\boldsymbol{y}} \left(\sum_{i=1}^{n} \log p(x_i \mid y_i) + \log p(y_i \mid y_{i-1}) \right) + \log p(stop \mid y_n) \\
&= \arg\max_{\boldsymbol{y}} \left(\sum_{i=1}^{n} w_{y_i \downarrow x_i} + w_{y_{i-1} \rightarrow y_i} \right) + w_{y_n \rightarrow stop} \\
&= \arg\max_{\boldsymbol{y}} \sum_{y,x} w_{y \downarrow x} freq(y \downarrow x; \boldsymbol{y}, \boldsymbol{x}) + \sum_{y,y'} w_{y \rightarrow y'} freq(y \rightarrow y'; \boldsymbol{y}) \\
&= \arg\max_{\boldsymbol{y}} \mathbf{w}^{\top} \mathbf{g}(\boldsymbol{x}, \boldsymbol{y})
\end{aligned}
$$

Restrictions #1, #2

1. The score function *factors locally*.
   – The more locally, the better!

$$score(\boldsymbol{x}, \boldsymbol{y}) = \sum_{C \in \mathcal{C}} \log \phi_C(\Pi_C(\boldsymbol{x}, \boldsymbol{y}))$$

2. The local scoring functions are linear in the features.

$$score(\boldsymbol{x}, \boldsymbol{y}) = \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y})$$

$$score(\boldsymbol{x}, \boldsymbol{y}) = \sum_{C \in \mathcal{C}} \mathbf{w}^\top \mathbf{f}(\Pi_C(\boldsymbol{x}, \boldsymbol{y}))$$

Running Example

		1	2	3	4	5	6	7	8	9	10
$\boldsymbol{x}$	=	Britain	sent	warships	across	the	English	Channel	Monday	to	rescue
$\boldsymbol{y}$	=	B	O	O	O	O	B	I	B	O	O
$\boldsymbol{y}'$	=	O	O	O	O	O	B	I	B	O	O

11	12	13	14	15	16	17	18	19	20
Britons	stranded	by	Eyjafjallajökull	's	volcanic	ash	cloud	.	
B	O	O	B	O	O	O	O	O	O
B	O	O	B	O	O	O	O	O	O

- BIO sequence labeling, here applied to NER
- Often solved with HMMs, CRFs, M^3Ns …

Factorization

$$prediction(\mathbf{x}, \mathbf{w}) = \underset{\mathbf{y} \in \mathcal{Y}(\mathbf{x})}{\arg\max}\, score(\mathbf{y}, \mathbf{w})$$

space of feasible outputs

Assumption:

$$score(\mathbf{y}, \mathbf{w}) = \mathbf{w}^\top \mathbf{f}(\mathbf{y}) = \sum_p \mathbf{w}^\top \mathbf{f}(\mathbf{y}_p)$$

Score is a sum of local “part” scores

Parts = nodes and edges in a graph, rules in a tree

Named Entity Recognition

$$f(x,y) = \sum_{(y_{i-1},y_i)\in y} f(y_{i-1},y_i) + \sum_{(x_i,y_i)} f(x_i,y_i)$$

$$f(y_{i-1},y_i)$$

$$y_{i-1} \quad y_i$$

B-ORG	I-ORG	O	B-ORG	I-ORG	I-ORG	O	O	I-LOC
Apple	Computer	bought	Smart	Systems	Inc.	located	in	Arkansas.

$$f(x_i,y_i)$$

$$x_i$$

Named Entity Recognition

$$f(x,y) = \sum_{(y_{i-1},y_i)\in y} f(y_{i-1},y_i) + \sum_{(x_i,y_i)} f(x_i,y_i)$$

$$f(y_{i-1},y_i)$$

$$y_{i-1} \quad y_i$$

B-ORG	I-ORG	O	B-ORG	I-ORG	I-ORG	O	O	I-LOC
Apple	Computer	bought	Smart	Systems	Inc.	located	in	Arkansas.

$$x_i$$

(What is *Not* A Linear Model?)

- Probabilistic models with hidden variables, requiring general MAP inference:

$$\arg\max_{\boldsymbol{y}} p(\boldsymbol{y} \mid \boldsymbol{x}) = \arg\max_{\boldsymbol{y}} \sum_{\boldsymbol{z}} p(\boldsymbol{y}, \boldsymbol{z} \mid \boldsymbol{x})$$

- Models based on non-linear kernels

$$\arg\max_{\boldsymbol{y}} \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y}) \quad = \quad \arg\max_{\boldsymbol{y}} \sum_{i=1}^{N} \alpha_i K\left(\langle \boldsymbol{x}_i, \boldsymbol{y}_i \rangle, \langle \boldsymbol{x}, \boldsymbol{y} \rangle\right)$$

- Most neural networks (those with non-linear activation functions)

Lecture Outline

- ✓ Viterbi algorithm
- ✓ Decoding more generally

3. Five views

1. Probabilistic Graphical Models

- View the linguistic structure as a collection of random variables that are interdependent.
- Represent interdependencies as a directed or undirected graphical model.
- Conditional probability tables (BNs) or factors (MNs) encode the probability distribution.
- Use standard techniques from PGMs to decode.

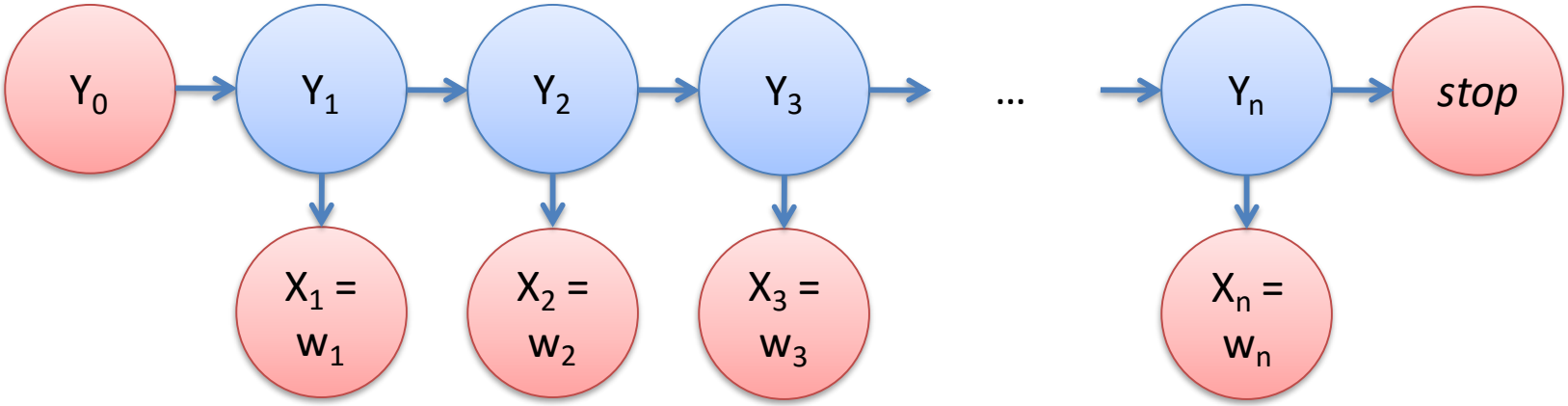
Inference in Graphical Models

- General algorithm for exact MPE inference: variable elimination.
  - Iteratively solve for the best values of each variable conditioned on values of "preceding" neighbors.
  - Then trace back.
  - Challenge: order the r.v.s for efficiency!

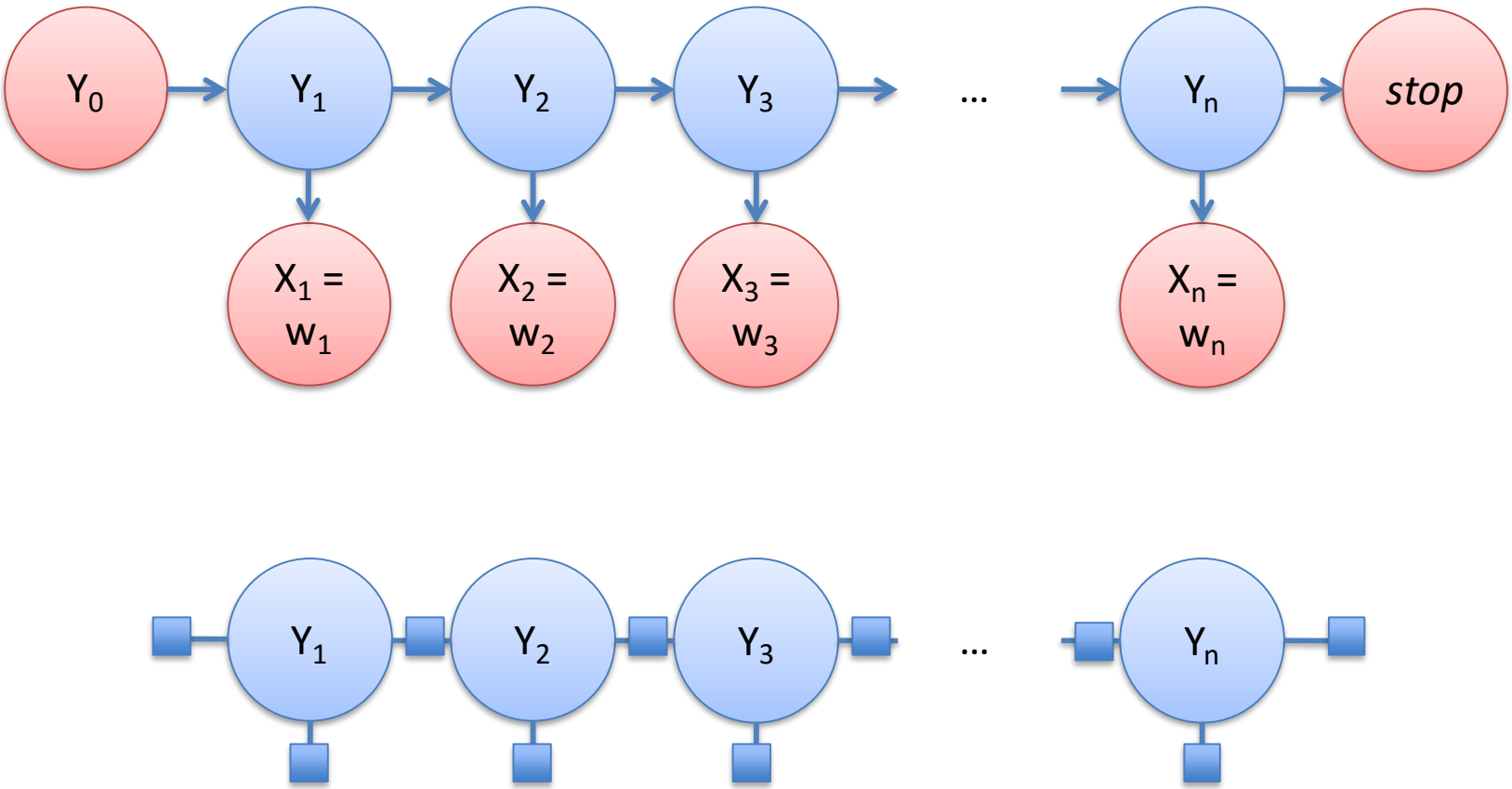
The Viterbi algorithm is an instance of max-product variable elimination!

Hidden Markov Model

- When we eliminate Y_1, we take a product of three relevant factors.
  - $\gamma(Y_1 \mid \textit{start})$
  - $\eta(w_1 \mid Y_1)$, reduced to the observed value w_1
  - $\gamma(Y_2 \mid Y_1)$

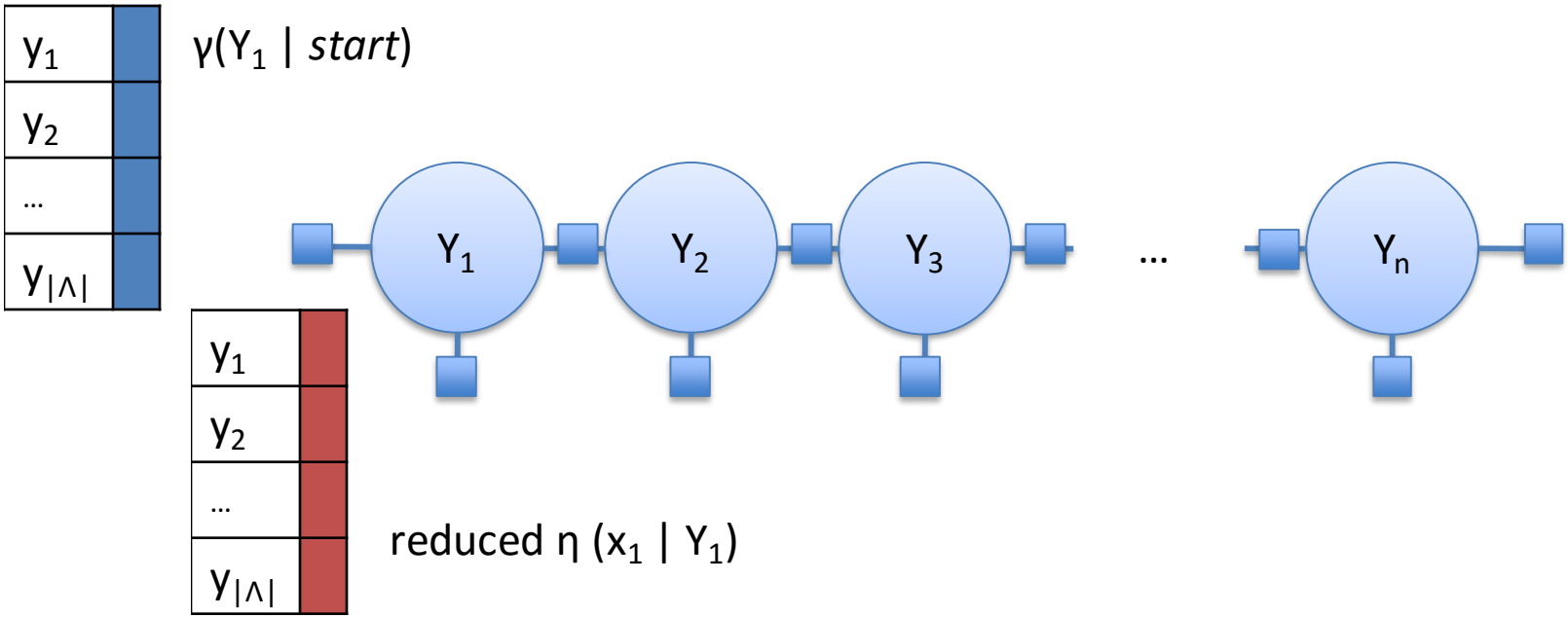


Factor Representation



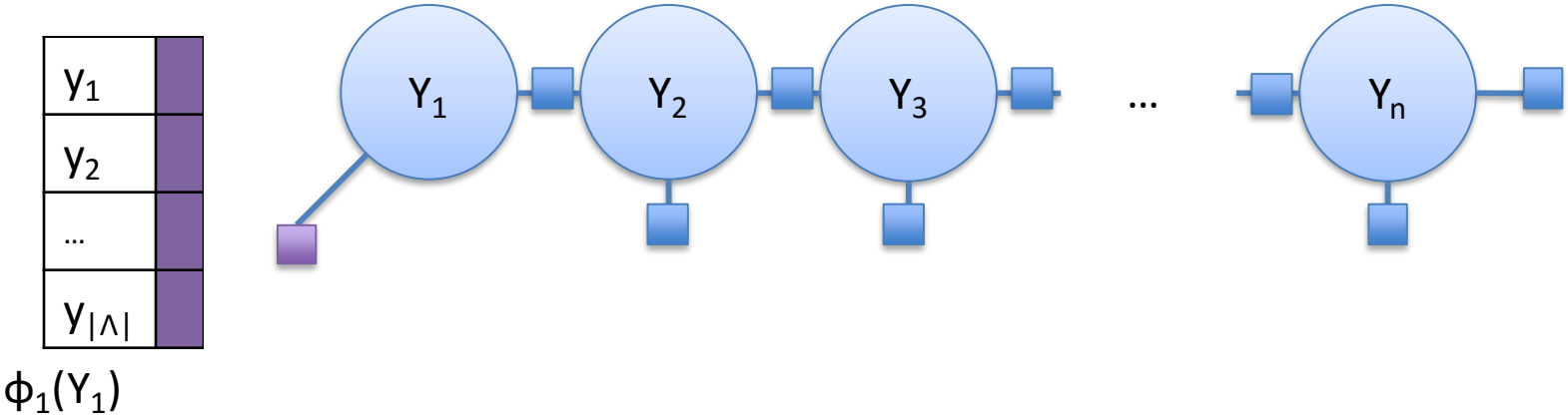
Hidden Markov Model

- When we eliminate Y_1, we first take a product of two factors that only involve Y_1.



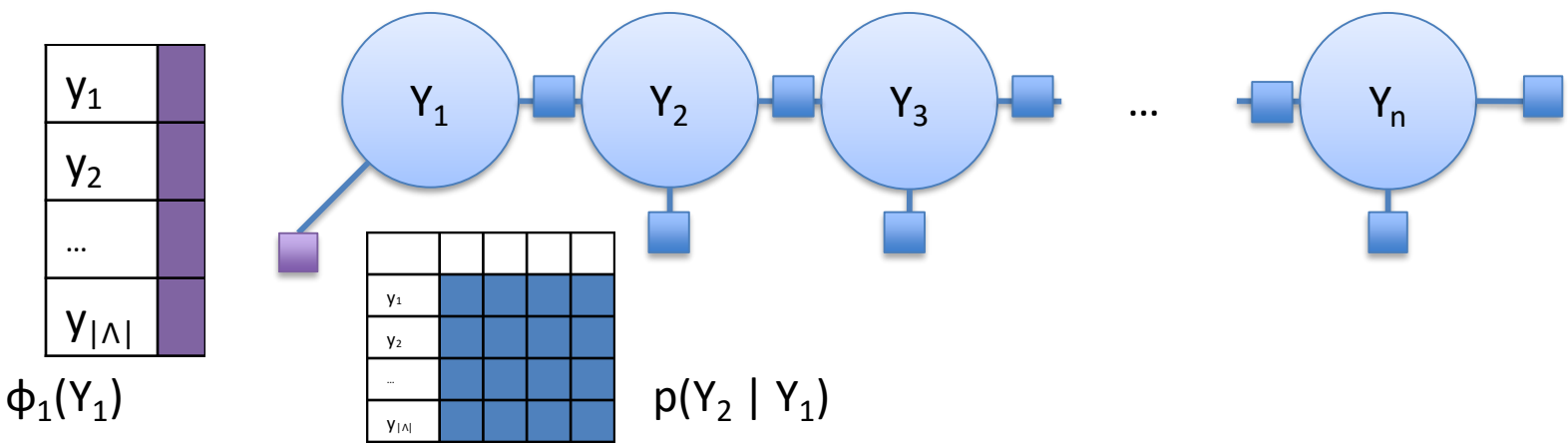
Hidden Markov Model

- When we eliminate Y_1, we first take a product of two factors that only involve Y_1.
- This is the Viterbi probability vector for Y_1.



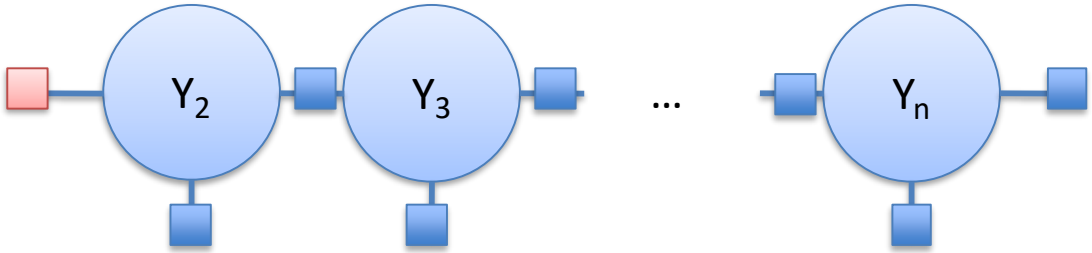
Hidden Markov Model

- When we eliminate Y_1, we first take a product of two factors that only involve Y_1.
- This is the Viterbi probability vector for Y_1.
- Eliminating Y_1 equates to solving the Viterbi probabilities for Y_2.



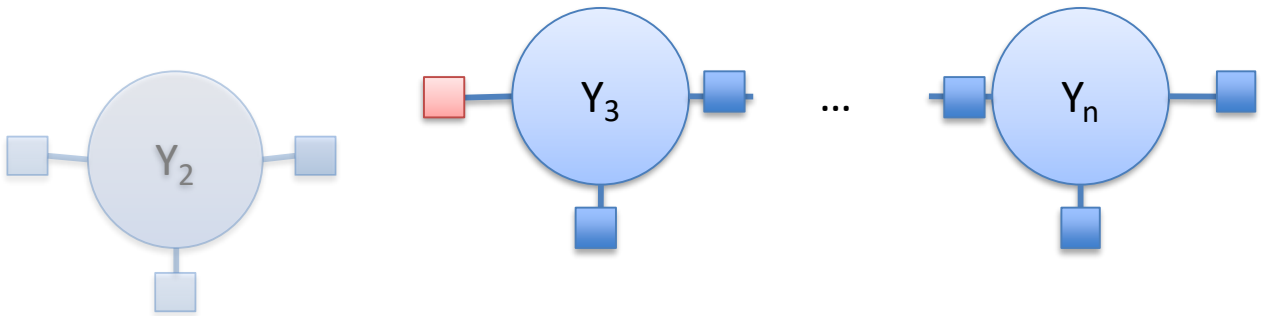
Hidden Markov Model

- Product of all factors involving Y_1, then reduce.
  - $\phi_2(Y_2) = \max_{y \in Val(Y_1)} (\phi_1(y) \times p(Y_2 \mid y))$
  - This factor holds Viterbi probabilities for Y_2.



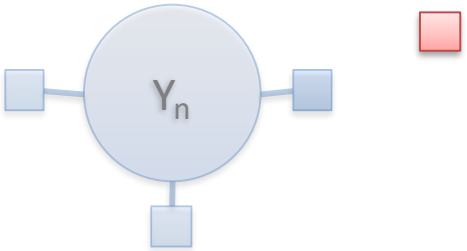
Hidden Markov Model

- When we eliminate Y_2, we take a product of the analogous two relevant factors.
- Then reduce.
  - $\phi_3(Y_3) = \max_{y \in Val(Y_2)} (\phi_2(y) \times p(Y_3 \mid y))$



Hidden Markov Model

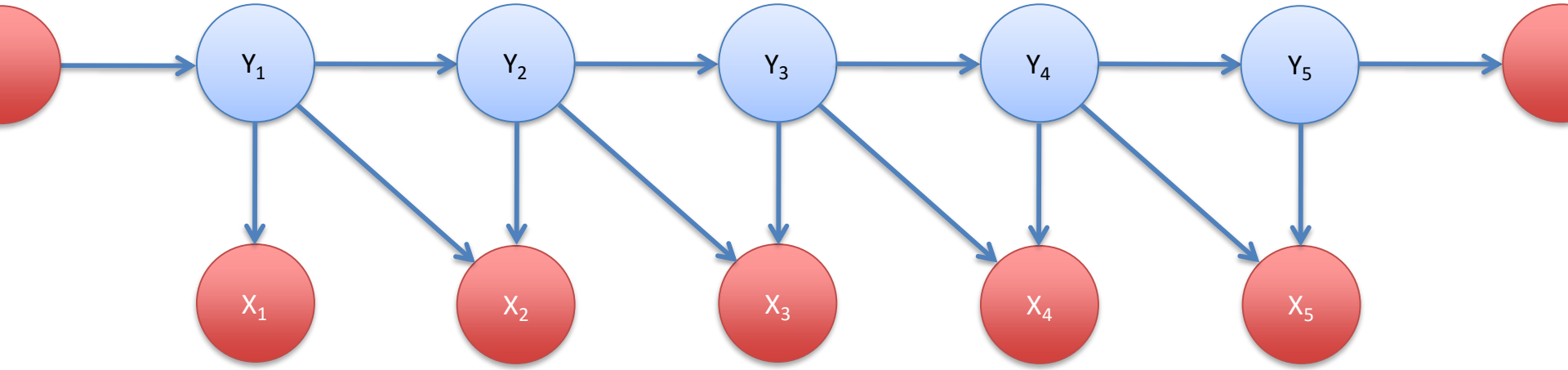
- At the end, we have one final factor with one row, ϕ_{n+1}.
- This is the score of the best sequence.
- Use backtrace to recover values.



Why Think This Way?

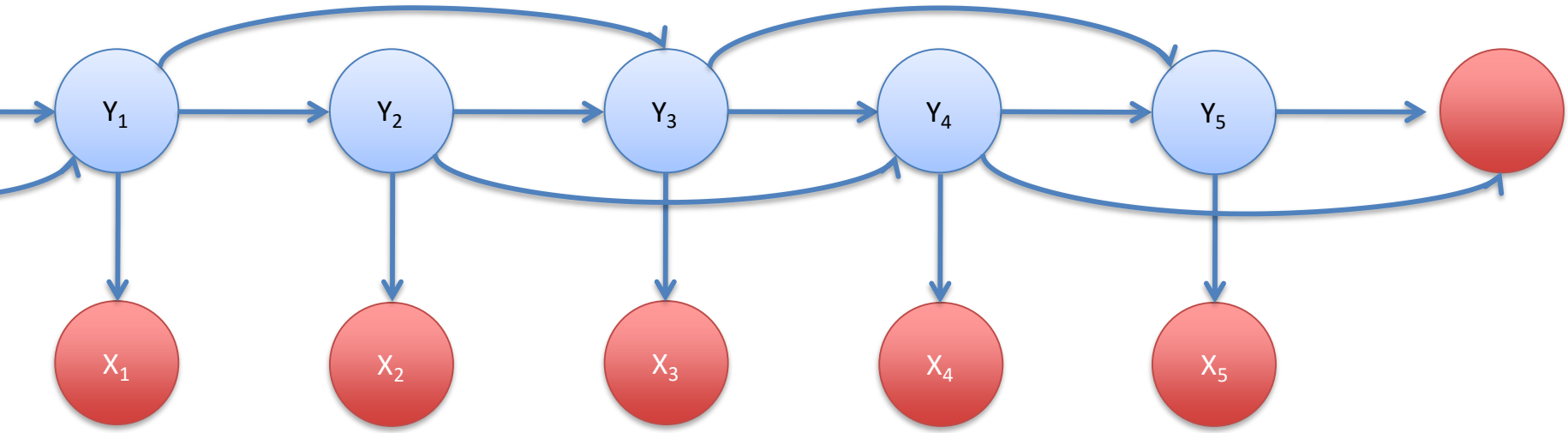
- Easy to see how to generalize HMMs.
  - More evidence
  - More factors
  - More hidden structure
  - More dependencies
- Probabilistic interpretation of factors is *not* central to finding the "best" **Y** …
  - Many factors are not conditional probability tables.

Generalization Example 1



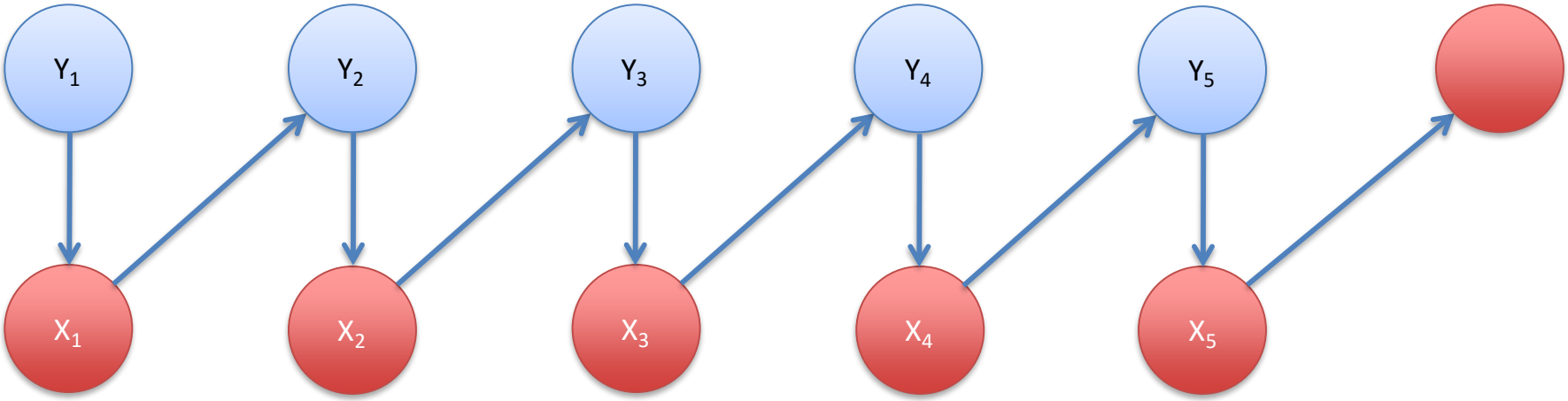
- Each word also depends on previous state.

Generalization Example 2



- “Trigram” HMM

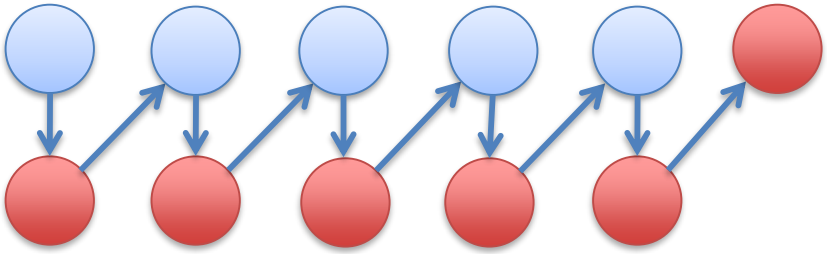
Generalization Example 3

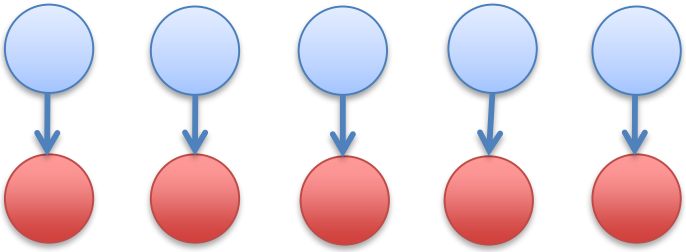


- Aggregate bigram model (Saul and Pereira, 1997)

Inference in Graphical Models

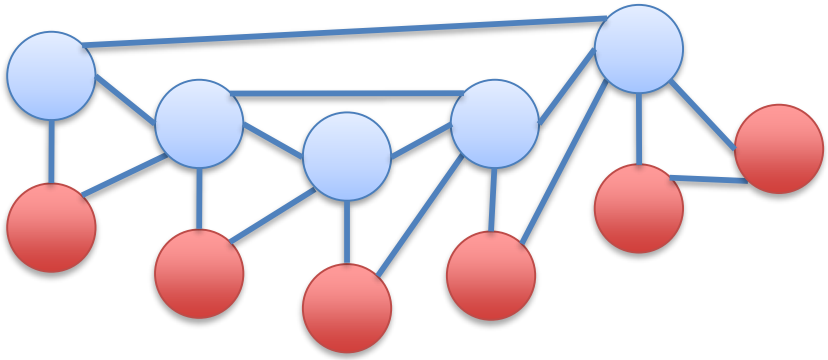
- Remember: more edges make inference more expensive.
  - Fewer edges means stronger independence.
- Really pleasant:

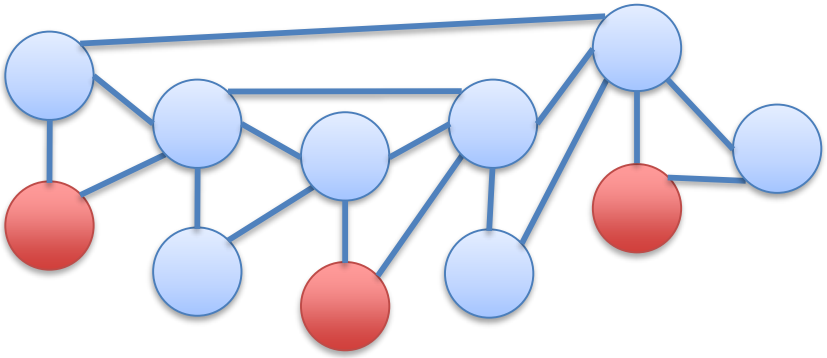




Inference in Graphical Models

- Remember: more edges make inference more expensive.
  - Fewer edges means stronger independence.
- Really unpleasant:





Lecture Outline

- ✓ Viterbi algorithm
- ✓ Decoding more generally

3. Five views
   - ✓ MPE/MAP inference in a graphical model

2. Polytopes

“Parts”

- Assume that feature function $\mathbf{g}$ breaks down into local parts.

$$\mathbf{g}(\boldsymbol{x}, \boldsymbol{y}) \quad = \quad \sum_{i=1}^{\#parts(\boldsymbol{x})} \mathbf{f}(\Pi_i(\boldsymbol{x}, \boldsymbol{y}))$$

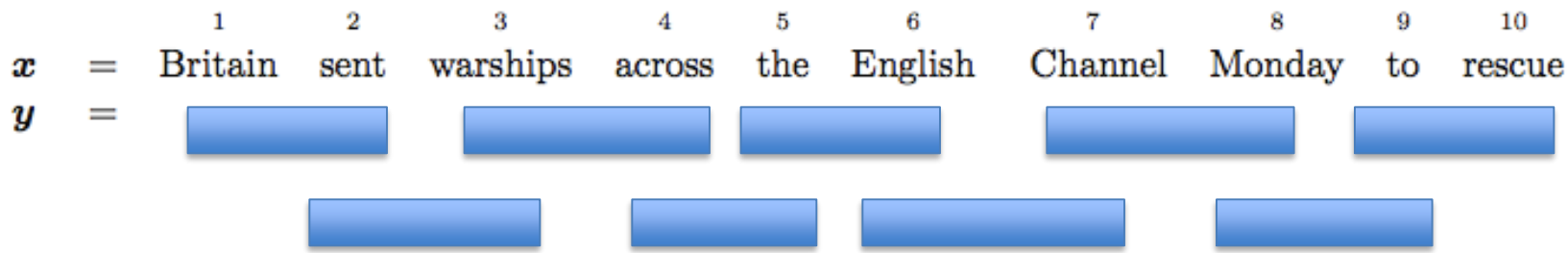
- Each part has an alphabet of possible values.
  - Decoding is choosing values for all parts, with consistency constraints.
  - (In the graphical models view, a part is a clique.)

Example

		1	2	3	4	5	6	7	8	9	10
$\boldsymbol{x}$	=	Britain	sent	warships	across	the	English	Channel	Monday	to	rescue
$\boldsymbol{y}$	=										

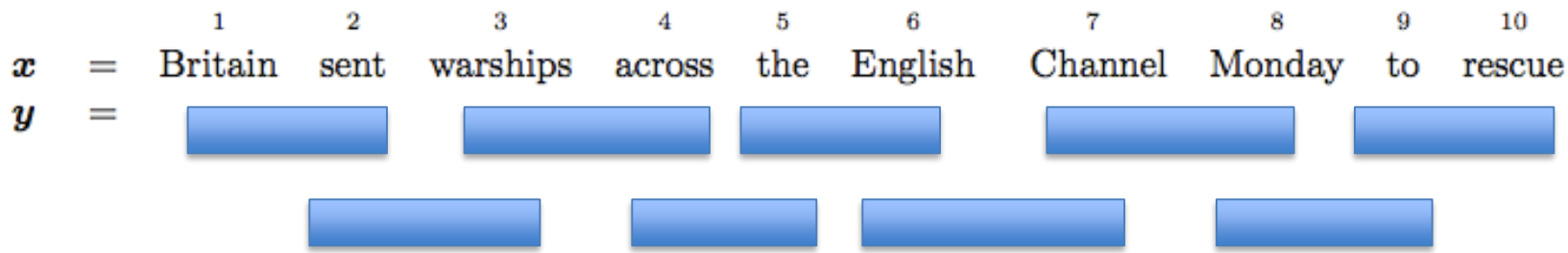
- One part per word, each is in {B, I, O}
- No features look at multiple parts
  - Fast inference
  - Not very expressive

Example



- One part per bigram, each is in {BB, BI, BO, IB, II, IO, OB, OO}
- Features and constraints can look at pairs
  - Slower inference
  - A bit more expressive

Geometric View



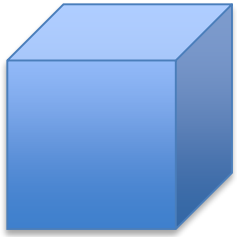
- Let $z_{i,\boldsymbol{\pi}}$ be 1 if part i takes value $\boldsymbol{\pi}$ and 0 otherwise.
- $\mathbf{z}$ is a vector in $\{0, 1\}^N$
  - N = total number of localized part values
  - Each $\mathbf{z}$ is a vertex of the unit cube

Score is Linear in **z**

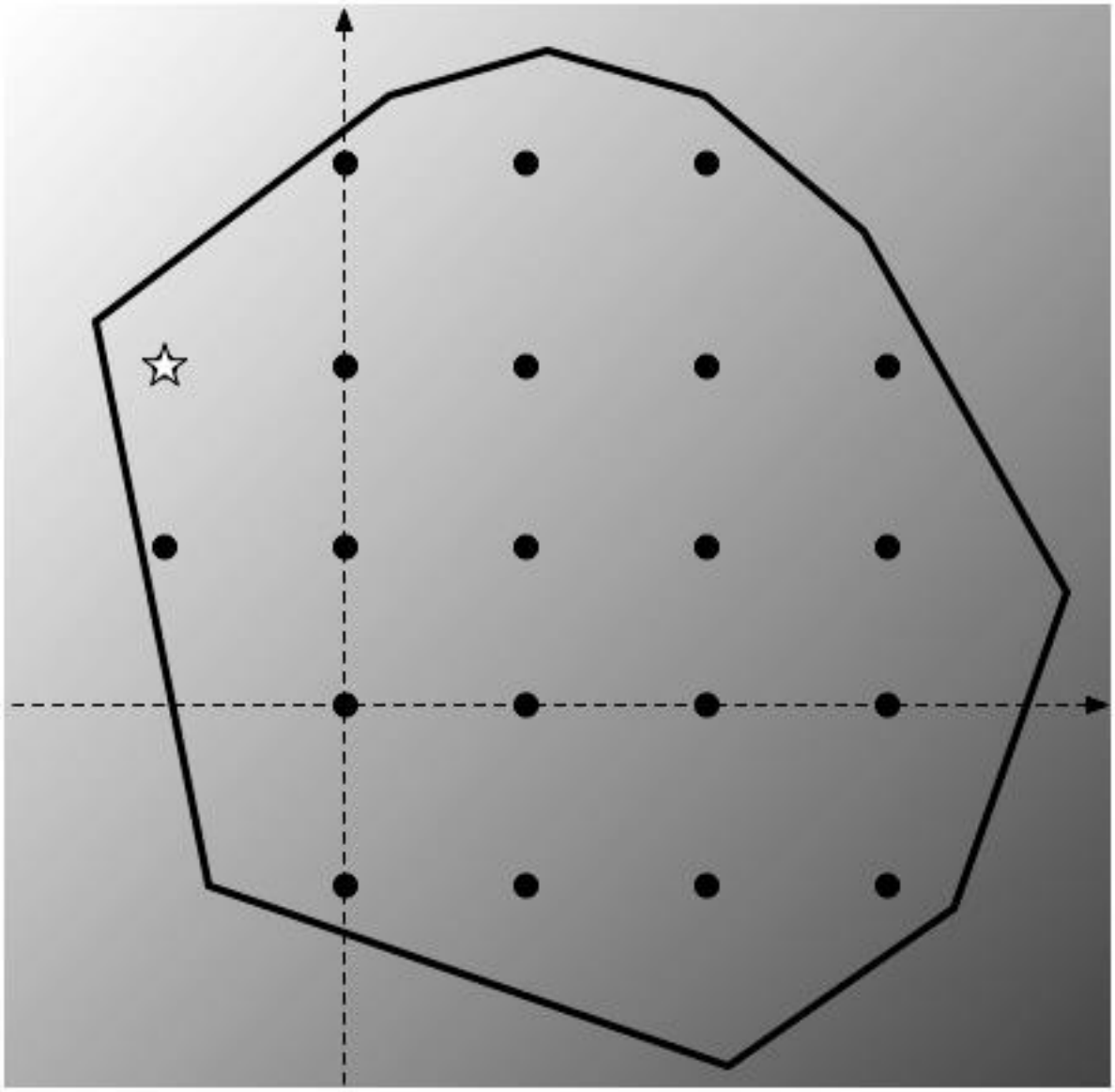
$$
\begin{aligned}
\arg\max_{\boldsymbol{y}} \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y}) &= \arg\max_{\boldsymbol{y}} \mathbf{w}^\top \sum_{i=1}^{\#parts(\boldsymbol{x})} \mathbf{f}(\Pi_i(\boldsymbol{x}, \boldsymbol{y})) \\
&= \arg\max_{\boldsymbol{y}} \mathbf{w}^\top \sum_{i=1}^{\#parts(\boldsymbol{x})} \sum_{\boldsymbol{\pi} \in \mathrm{Values}(\Pi_i)} \mathbf{f}(\boldsymbol{\pi}) \mathbf{1}\{\Pi_i(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{\pi}\} \\
&= \arg\max_{\mathbf{z} \in \mathcal{Z}_{\boldsymbol{x}}} \mathbf{w}^\top \sum_{i=1}^{\#parts(\boldsymbol{x})} \sum_{\boldsymbol{\pi} \in \mathrm{Values}(\Pi_i)} \mathbf{f}(\boldsymbol{\pi}) z_{i,\boldsymbol{\pi}} \\
&= \arg\max_{\mathbf{z} \in \mathcal{Z}_{\boldsymbol{x}}} \mathbf{w}^\top \mathbf{F}_{\boldsymbol{x}} \mathbf{z} \\
&= \arg\max_{\mathbf{z} \in \mathcal{Z}_{\boldsymbol{x}}} \left(\mathbf{w}^\top \mathbf{F}_{\boldsymbol{x}}\right) \mathbf{z}
\end{aligned}
$$

not really equal; need to transform back to get **y**

Polyhedra

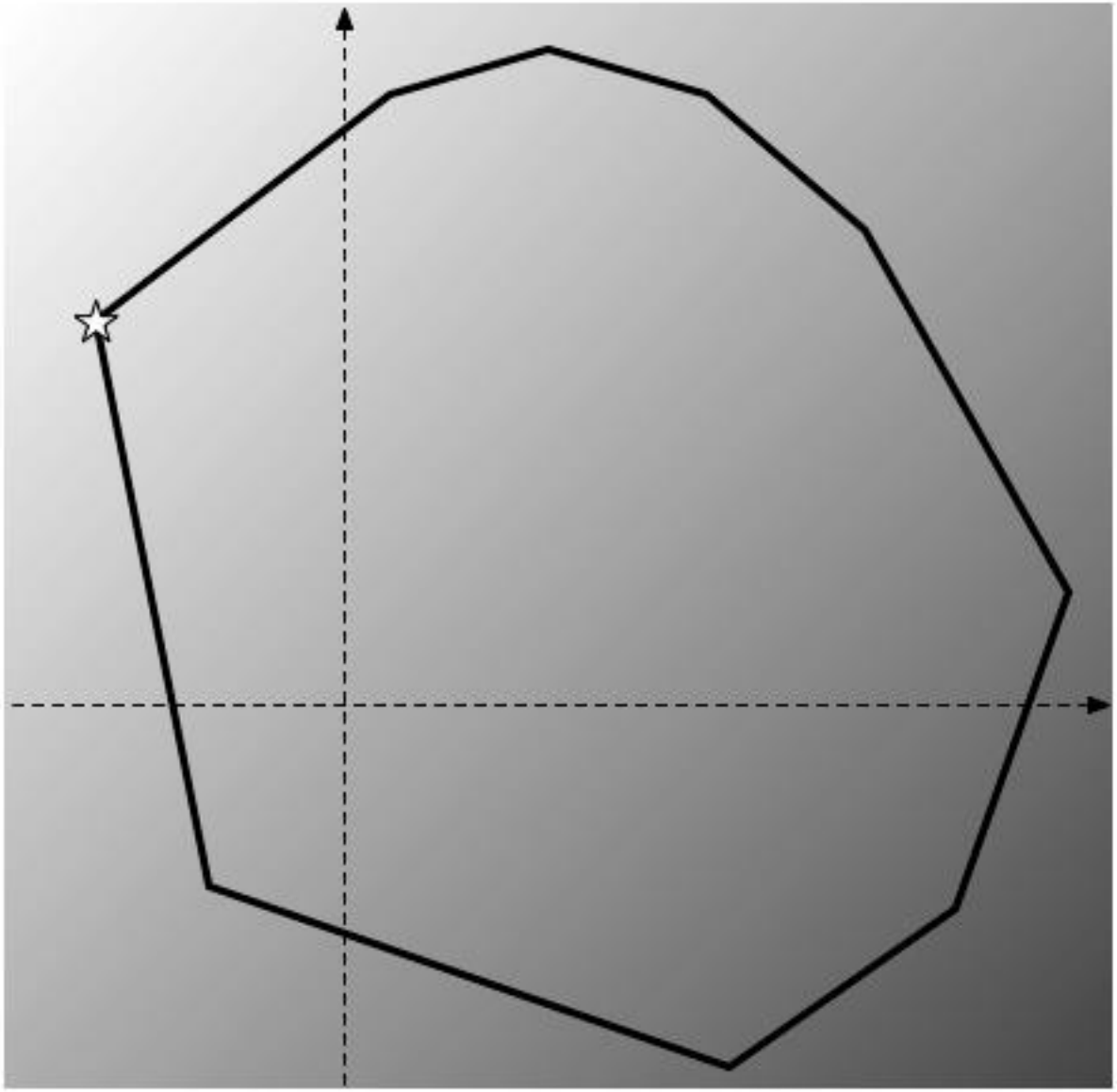


- Not all vertices of the *N*-dimensional unit cube satisfy the constraints.
  - E.g., can't have $z_{1,\mathrm{BI}} = 1$ and $z_{2,\mathrm{BI}} = 1$
- Sometimes we can write down a small (polynomial number) of linear constraints on **z**.
- Result: linear objective, linear constraints, integer constraints …



Integer Linear Programming

- Very easy to add new constraints and non-local features.
- Many decoding problems have been mapped to ILP (sequence labeling, parsing, …), but it's *not* always trivial.
- NP-hard in general.
  - But there are packages that often work well in practice (e.g., CPLEX)
  - Specialized algorithms in some cases
  - LP relaxation for approximate solutions



Remark

- Graphical models assumed a probabilistic interpretation
  - Though they are not always learned using a probabilistic interpretation!

- The polytope view is agnostic about how you interpret the weights.
  - It only says that the decoding problem is an ILP.

3. Weighted Parsing

Grammars

- Grammars are often associated with natural language parsing, but they are extremely powerful for imposing constraints.
- We can add weights to them.
  - HMMs are a kind of weighted regular grammar (closely connected to WFSAs)
  - PCFGs are a kind of weighted CFG
  - Many, many more.
- Weighted parsing: find the maximum-weighted derivation for a string **x**.

Decoding as Weighted Parsing

- Every valid **y** is a grammatical derivation (parse) for **x**.
  - HMM: sequence of "grammatical" states is one allowed by the transition table.
- Augment parsing algorithms with weights and find the best parse.

The Viterbi algorithm is an instance of recognition by a weighted grammar!

BIO Tagging as a CFG

$$
\begin{array}{lllllllllll}
N_{\Rightarrow} & \rightarrow & B\ R_B & R_B & \rightarrow & B\ R_B & R_I & \rightarrow & B\ R_B & R_O & \rightarrow & B\ R_B \\
N_{\Rightarrow} & \rightarrow & O\ R_O & R_B & \rightarrow & O\ R_O & R_I & \rightarrow & O\ R_O & R_O & \rightarrow & O\ R_O \\
 & & & R_B & \rightarrow & I\ R_I & R_I & \rightarrow & I\ R_I & & & \\
 & & & R_B & \rightarrow & \epsilon & R_I & \rightarrow & \epsilon & R_O & \rightarrow & \epsilon \\
 & & & & & & & & & & & \\
 & \forall x \in \Sigma, & & B & \rightarrow & x & I & \rightarrow & x & O & \rightarrow & x
\end{array}
$$

- Weighted (or probabilistic) CKY is a dynamic programming algorithm very similar in structure to classical CKY.

4. Paths and Hyperpaths

Best Path

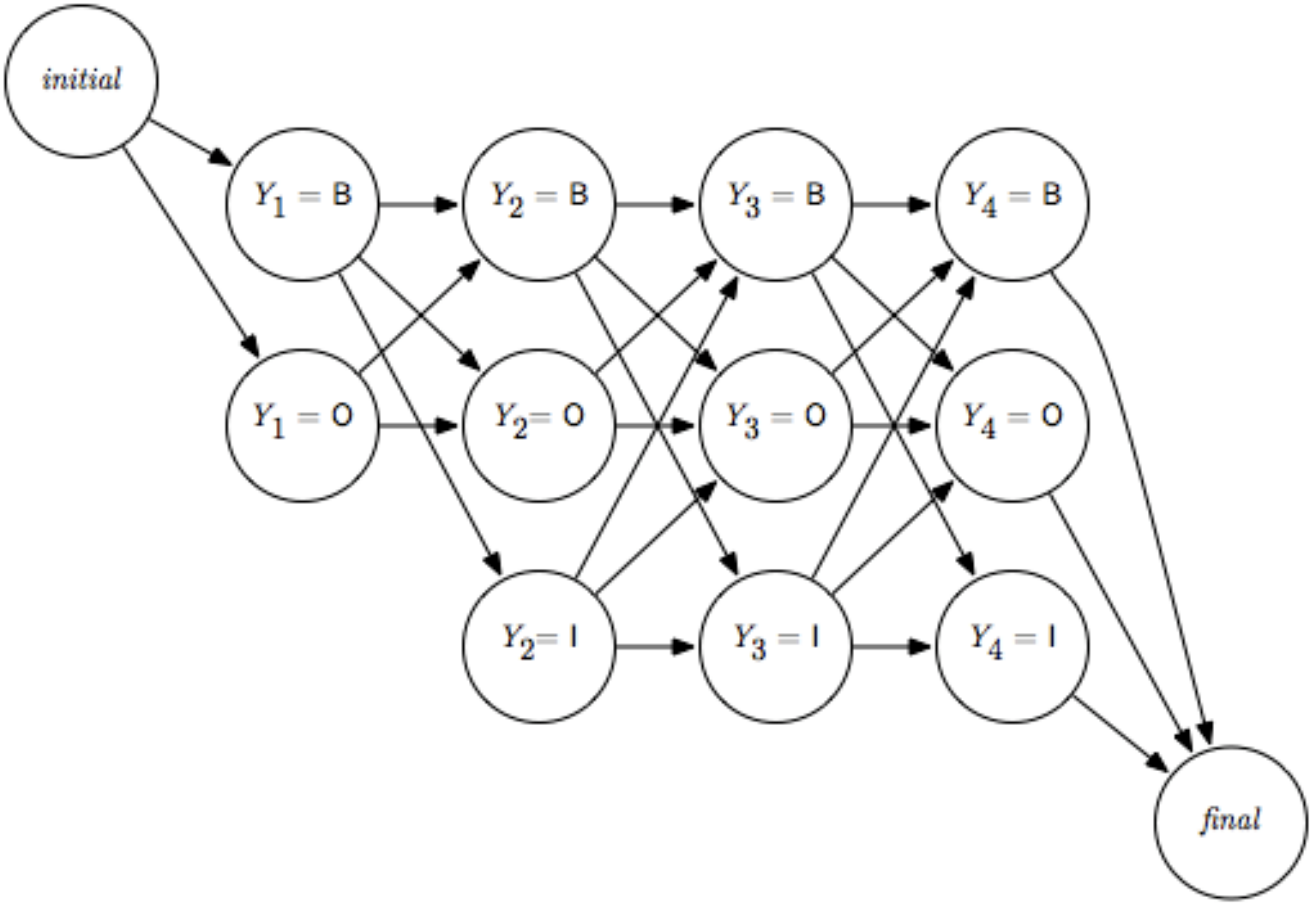
- General idea: take ***x*** and build a graph.
- Score of a path factors into the edges.

$$\arg\max_{\boldsymbol{y}} \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y}) \quad = \quad \arg\max_{\boldsymbol{y}} \mathbf{w}^\top \sum_{e \in \text{Edges}} \mathbf{f}(e) \mathbf{1}\{e \text{ is crossed by } \boldsymbol{y}\text{'s path}\}$$

- Decoding is finding the *best* path.

The Viterbi algorithm is an instance of finding a best path!

“Lattice” View of Viterbi



A Generic Best Path Algorithm

- Input: directed graph $G = (V, E)$, cost : $E \to \mathbb{R}$, start vertex v_0
- Output: $d : V \to \mathbb{R}$ (shortest path function) and back pointers $b : V \to V$

for all $v \in V \setminus \{v_0\}$, $d(v) := \infty$ and $b(v) := \emptyset$
set $d(v_0) = 0$
while d has not converged:
    pick an arbitrary edge (u, v)
    if $d(u) + cost(u, v) < d(v)$:
      $d(v) := d(u) + cost(u, v)$
      $b(v) := u$

Ordering Updates

- Naïve ways of choosing edges will lead to cyclic updating and gross inefficiency!
- Before considering various ways of doing it, let's consider how the Viterbi algorithm is essentially solving the same problem.

Viterbi Algorithm
(In the Style of A Best Path Algorithm)

- Input:
  - directed graph $G = (V, E)$ where
    each vertex $v = (q, t)$, $q \in Q \cup \{\emptyset\}$, $t \in \{-1, 0, 1, \ldots, n\}$
    and each edge $(u, v) = ((q, t), (q', t + 1))$
  - cost : $E \to \mathbb{R}$, defined by
    $\text{cost}((q, t), (q', t + 1)) = -\log \gamma(q' \mid q) - \log \eta(s_{t+1} \mid q) - \log (1 - \xi(q))$
    $\text{cost}((q, n - 1), (q', n)) = -\log \gamma(q' \mid q) - \log \eta(s_{t+1} \mid q) - \log \xi(q')$
    $\text{cost}((\emptyset, -1), (q, 0)) = -\log \pi(q)$
  - fixed start vertex $v_0 = (\emptyset, -1)$
- Output: $d : V \to \mathbb{R}$ (shortest path function) and back pointers $b : V \to V$

for all $v \in V \setminus \{v_0\}$, $d(v) := \infty$ and $b(v) := \emptyset$
set $d(v_0) = 0$
perform a topological sort on V
~~while d has not converged:~~ for each v in top-sort order:
    ~~pick an arbitrary edge (u, v)~~
    for each $(u, v) \in E$:
      if $d(u) + \text{cost}(u, v) < d(v)$:
          $d(v) := d(u) + \text{cost}(u, v)$
          $b(v) := u$
    // d(v) and b(v) are now known

The Viterbi Trick

- From a “best path” perspective, Viterbi is:
  - defining the vertices and edges to have special structure (state/time step)
  - assigning costs based on HMM weights and the specific input string $s_1 \ldots s_n$
  - ordering the edges cleverly to make things efficient
- Note also: Viterbi's graph has no cycles.

Another Variant: "Forward" Updating

- After topological sort, can also choose all edges going *out of* current node.

```
for all v ∈ V \ {v0}, d(v) := ∞ and b(v) := ∅
set d(v0) = 0
perform a topological sort on V
for each u in top-sort order:
    for each (u, v) ∈ E:
     if d(u) + cost(u, v) < d(v):
          d(v) := d(u) + cost(u, v)
          b(v) := u
```

Memoized Recursion

- Input: directed graph $G = (V, E)$, cost : $E \to \mathbb{R}$, start vertex v_0, target vertex v_t
- Output: $d : V \to \mathbb{R}$ (shortest path function) and back pointers $b : V \to V$

```
for all v ∈ V \ {v0}, d(v) := ∅ and b(v) := ∅
set d(v0) = 0
memoize(vt)

memoize(v):
    // guaranteed to return best-cost path score for v
    if d(v) = ∅:
      d(v) := ∞
      for each (u, v) ∈ E:
            if memoize(u) + cost(u, v) < d(v):
                  d(v) := d(u) + cost(u, v)
                  b(v) := u
    return d(v)
```

A Generic Best Path Algorithm

- Input: directed graph $G = (V, E)$, cost : $E \to \mathbb{R}$, start vertex v_0
- Output: $d : V \to \mathbb{R}$ (shortest path function) and back pointers $b : V \to V$

for all $v \in V \setminus \{v_0\}$, $d(v) := \infty$ and $b(v) := \emptyset$
set $d(v_0) = 0$
while d has not converged:
    pick an arbitrary edge (u, v)
    if $d(u) + cost(u, v) < d(v)$:
      $d(v) := d(u) + cost(u, v)$
      $b(v) := u$

Dijkstra's Algorithm

- Input: directed graph $G = (V, E)$, cost : $E \to \mathbb{R}_{\geq 0}$ (important!), start vertex v_0
- Output: $d : V \to \mathbb{R}$ (shortest path function) and back pointers $b : V \to V$

for all $v \in V \setminus \{v_0\}$, $d(v) := \infty$ and $b(v) := \emptyset$
set $d(v_0) = 0$
Q := priority queue on V ordered by d (lower cost = higher priority)
~~while d has not converged:~~ while Q is not empty:
    ~~pick an arbitrary edge (u, v)~~
    u := extract-min(Q)
    for each $(u, v) \in E$:
      if $d(u) + cost(u, v) < d(v)$:
            $d(v) := d(u) + cost(u, v)$
            $b(v) := u$
            update v's priority in Q

A* Algorithm

- Input: directed graph $G = (V, E)$, cost : $E \to \mathbb{R}_{\geq 0}$, start vertex v_0, target vertex v_t, heuristic $h : V \to \mathbb{R}_{\geq 0}$ such that $h(v) \leq \text{best-cost}(v, v_t)$
- Output: $d : V \to \mathbb{R}$ (shortest path function) and back pointers $b : V \to V$

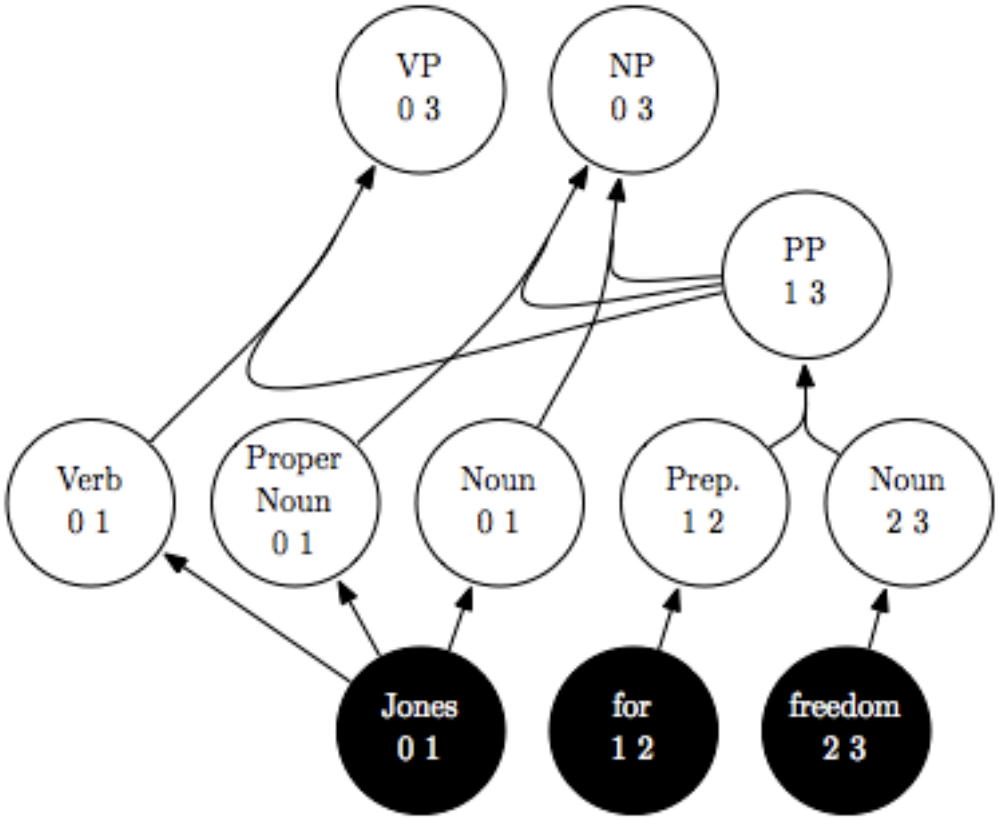
```
for all v ∈ V \ {v0}, d(v) := ∞ and b(v) := ∅
set d(v0) = 0
Q := priority queue on V ordered by d + h (lower cost = higher priority)
while Q is not empty:
    u := extract-min(Q)
    for each (u, v) ∈ E:
      if d(u) + cost(u, v) < d(v):
            d(v) := d(u) + cost(u, v)
            b(v) := u
            update v's priority in Q
```

Minimum Cost Hyperpath

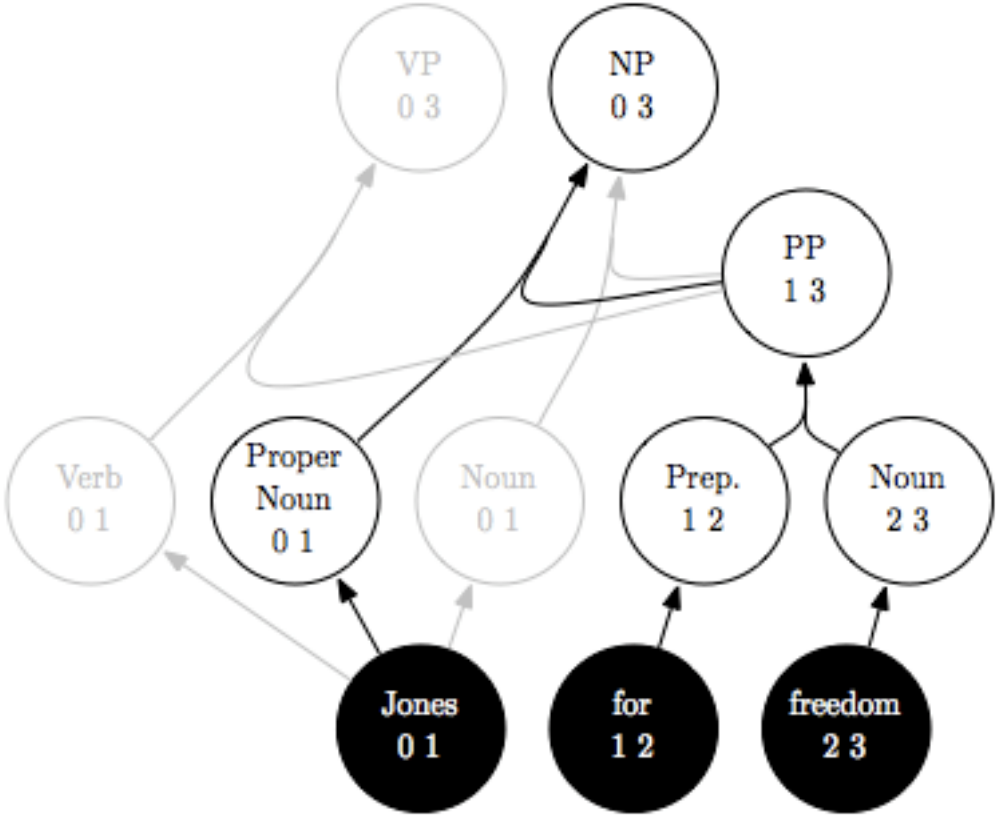
- General idea: take ***x*** and build a hypergraph.
- Score of a hyperpath factors into the hyperedges.
- Decoding is finding the best *hyperpath*.

- This connection was elucidated by Klein and Manning (2002).

Parsing as a Hypergraph

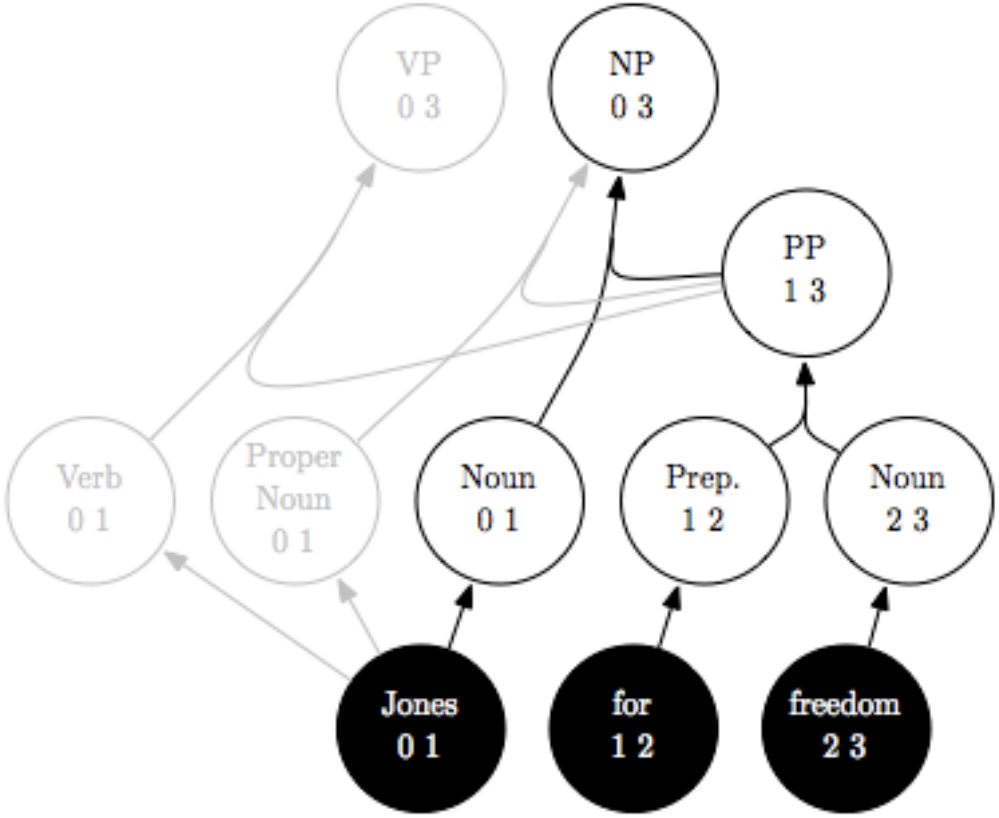


Parsing as a Hypergraph



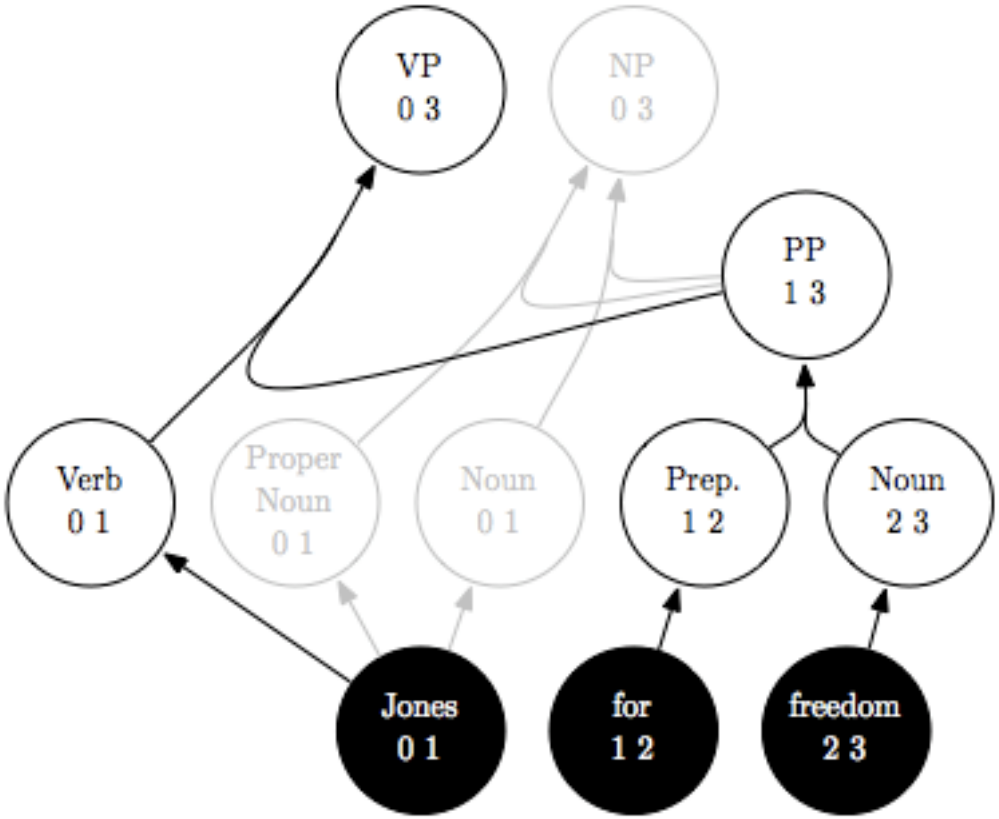
cf. “Dean for democracy”

Parsing as a Hypergraph



Forced to work on his thesis, sunshine streaming in the window, Mike experienced a …

Parsing as a Hypergraph



Forced to work on his thesis, sunshine streaming in the window, Mike began to …

Why Hypergraphs?

- Useful, compact encoding of the hypothesis space.
  - Build hypothesis space using local features, maybe do some filtering.
  - Pass it off to another module for more fine-grained scoring with richer or more expensive features.

5. Weighted Logic Programming

Logic Programming

- Start with a set of axioms and a set of inference rules.

$$\forall A, C, \quad \text{ancestor}(A, C) \Leftarrow \text{parent}(A, C)$$
$$\forall A, C, \quad \text{ancestor}(A, C) \Leftarrow \bigvee_B \text{ancestor}(A, B) \wedge \text{parent}(B, C)$$

- The goal is to prove a specific theorem, goal.
- Many approaches, but we assume a *deductive* approach.
  - Start with axioms, iteratively produce more theorems.

$$
\begin{array}{ll}
 & \text{label-bigram}(\text{“B”}, \text{“B”}) \\
 & \text{label-bigram}(\text{“B”}, \text{“I”}) \\
 & \text{label-bigram}(\text{“B”}, \text{“O”}) \\
 & \text{label-bigram}(\text{“I”}, \text{“B”}) \\
 & \text{label-bigram}(\text{“I”}, \text{“I”}) \\
 & \text{label-bigram}(\text{“I”}, \text{“O”}) \\
 & \text{label-bigram}(\text{“O”}, \text{“B”}) \\
 & \text{label-bigram}(\text{“O”}, \text{“O”}) \\
\forall x \in \Sigma, & \text{labeled-word}(x, \text{“B”}) \\
\forall x \in \Sigma, & \text{labeled-word}(x, \text{“I”}) \\
\forall x \in \Sigma, & \text{labeled-word}(x, \text{“O”})
\end{array}
$$

$$
\begin{aligned}
\forall \ell \in \Lambda, \quad \mathsf{v}(\ell, 1) &= \text{labeled-word}(x_1, \ell) \\
\forall \ell \in \Lambda, \quad \mathsf{v}(\ell, i) &= \bigvee_{\ell' \in \Lambda} \mathsf{v}(\ell', i-1) \wedge \text{label-bigram}(\ell', \ell) \wedge \text{labeled-word}(x_i, \ell) \\
\text{goal} &= \bigvee_{\ell \in \Lambda} \mathsf{v}(\ell, n)
\end{aligned}
$$

Weighted Logic Programming

- Twist: axioms have weights.
- Want the proof of goal with the best score:

$$\arg\max_{\boldsymbol{y}} \mathbf{w}^\top \mathbf{g}(\boldsymbol{x}, \boldsymbol{y}) \quad = \quad \arg\max_{\boldsymbol{y}} \mathbf{w}^\top \sum_{a \in \mathrm{Axioms}} \mathbf{f}(a) \mathit{freq}(a; \boldsymbol{y})$$

- Note that axioms can be used more than once in a proof (**y**).

Whence WLP?

- Shieber, Schabes, and Pereira (1995): many parsing algorithms can be understood in the same deductive logic framework.
- Goodman (1999): add weights in a semiring, get many useful NLP algorithms.
- Eisner, Goldlust, and Smith (2004, 2005): semiring-generic algorithms, Dyna.

Dynamic Programming

- Most views (exception is polytopes) can be understood as DP algorithms.
  - The low-level *procedures* we use are often DP.
  - Even DP is too high-level to know the best way to implement.
- Break a problem into slightly smaller problems with **optimal substructure**.
  - Best path to v depends on best paths to all u such that $(u,v) \in E$.
- **Overlapping subproblems**: each subproblem gets used repeatedly, and there aren't too many of them.

Dynamic Programming

- Three main strategies for DP:
  - Viterbi, Levenshtein edit distance, CKY: predefined, "clever" ordering.
  - Memoization
  - Agenda (Dijkstra's algorithm, A*)
- Things to remember in general:
  - The hypergraph may too big to represent explicitly; exhaustive calculation may be too expensive.
  - The hypergraph may or may not have properties that make "clever" orderings possible.
  - DP does *not* imply polynomial time and space! Most common approximations when the desired state space is too big: beam search, cube pruning, agendas with early stopping, …

Summary

- Decoding is the general problem of choosing a complex structure.
  - Linguistic analysis, machine translation, speech recognition, …
  - Statistical models are usually involved (not necessarily probabilistic).
- No perfect general view, but much can be gained through a combination of views.
- First question: can I solve it exactly with DP?