Learning and parsing stochastic unification-based grammars

Mark Johnson

Brown University

(BLLIP)

COLT'03 talk

Joint work with Stuart Geman and Stefan Riezler

Supported by NSF grants LIS 9720368 and IIS0095940

Talk outline

- Lexical Functional Grammars
- Stochastic Lexical Functional Grammars
- Supervised training from parsed corpora
- Semi-supervised training from partially labelled data
- Dynamic programming using MRF graphical models

“Big picture” issues

- Human language can be understood in terms of knowledge that can be used in many ways
- *What kinds of knowledge are involved in the use of human language?*
  - *Linguistic knowledge* about the surface form $\leftrightarrow$ “meaning” relationship
  - This relationship involves complex *hidden syntactic structure* (described by a *grammar*)
  - *World knowledge* and *pragmatic knowledge* are also involved

“Big picture” issues (cont.)

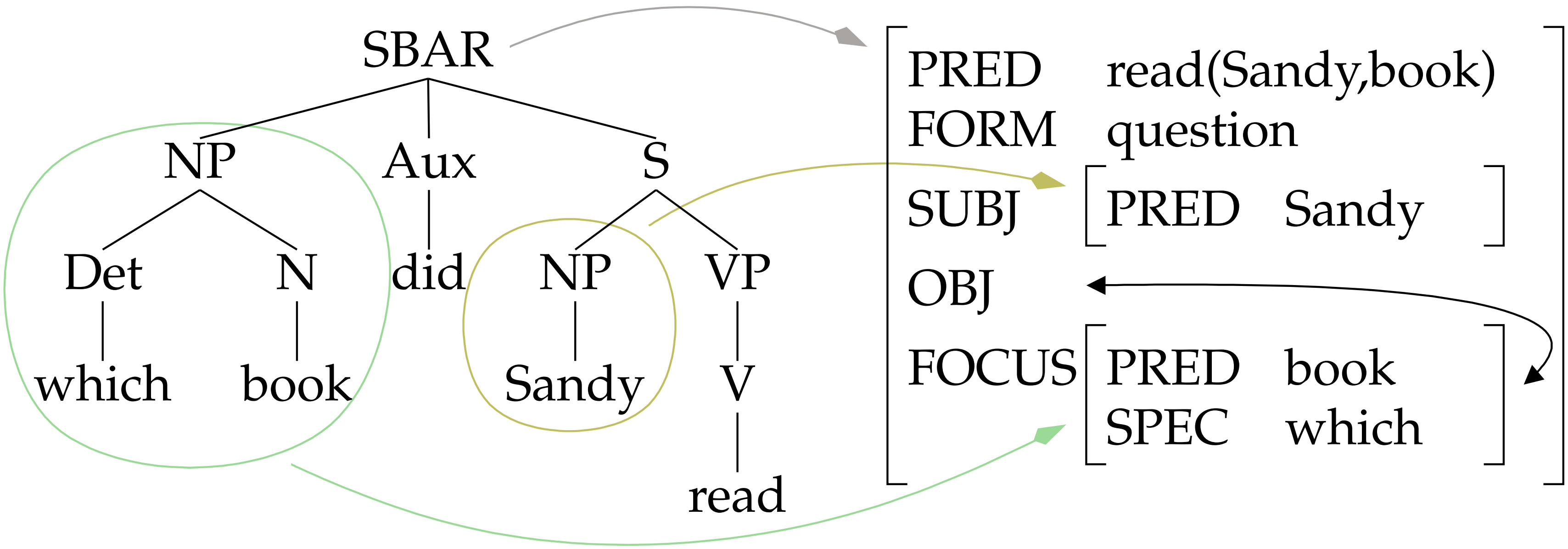
- *Which components of this knowledge are innate, and which are learnt?*
  - The phonological components (sounds) of words are arbitrary $\Rightarrow$ relationship between phonological representations and lexical meanings must be learnt (*how?*)
  - The (abstract) syntactic structure of most languages seems very similar, and no one knows how it might be learnt $\Rightarrow$ perhaps syntax is innate?
- How exactly is the knowledge learnt? (Can we learn it explicitly?)
- How is all this knowledge used in production and comprehension?

Stochastic Lexical-Functional Grammar

- A Lexical-Functional Grammar (LFG) models the relationship between phonological, syntactic and semantic structures that together form the *parse* of a sentence
- An LFG defines the set $\mathcal{Y}$ of parses possible in a language
- Most sentences are ambiguous (1 to 10,000 parses)
- A Stochastic LFG defines a (conditional) probability distribution over $\mathcal{Y}$
- Organization of an SLFG:
  - *manually specify* possible parses using an LFG
  - *manually specify* conditioning features
  - *learn* feature weights from training data

Q: What is the most effective way to describe a human language? (grammar, corpus)

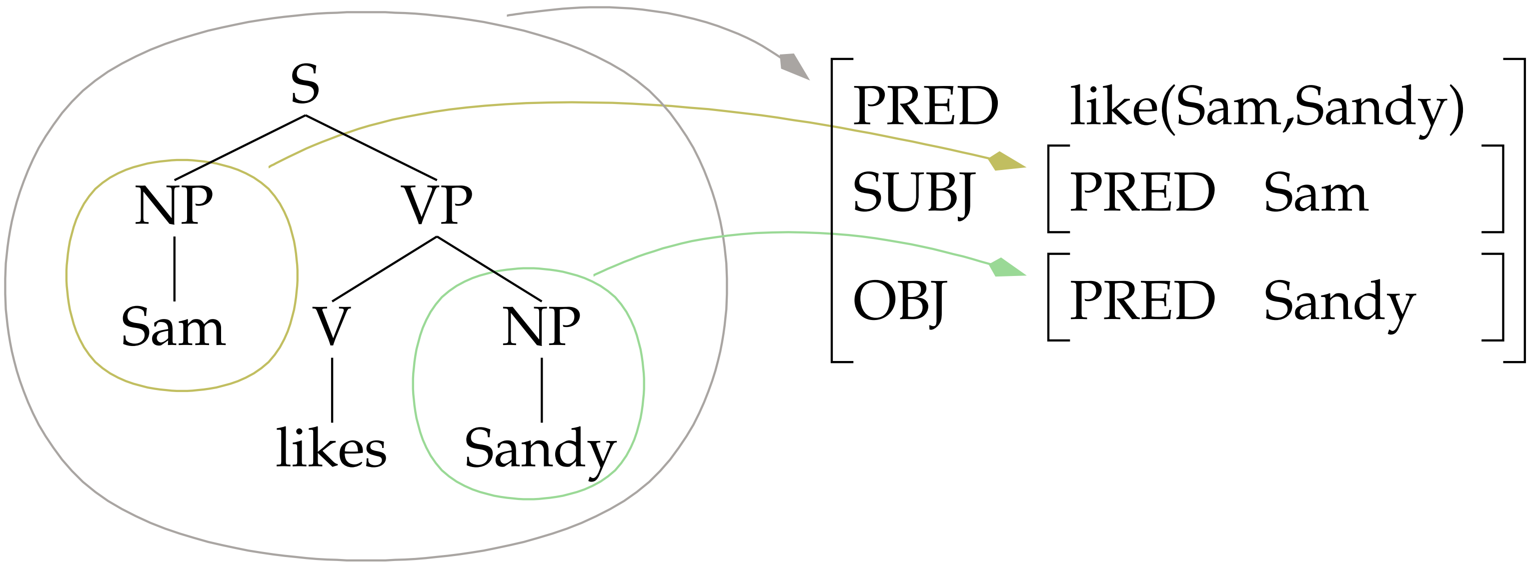
Representations of Lexical-Functional Grammar



c(onstituent) structure *f(unctional) structure*

- Not depicted here: morphological structure, semantic structure
- This work focuses on disambiguating c-structure and f-structure
  - Almost all c and f-structure ambiguity is reflected in semantic structure ambiguity
  - Other semantic structure ambiguity (e.g., quantifier scope) is very hard for humans to disambiguate

How an LFG describes parses



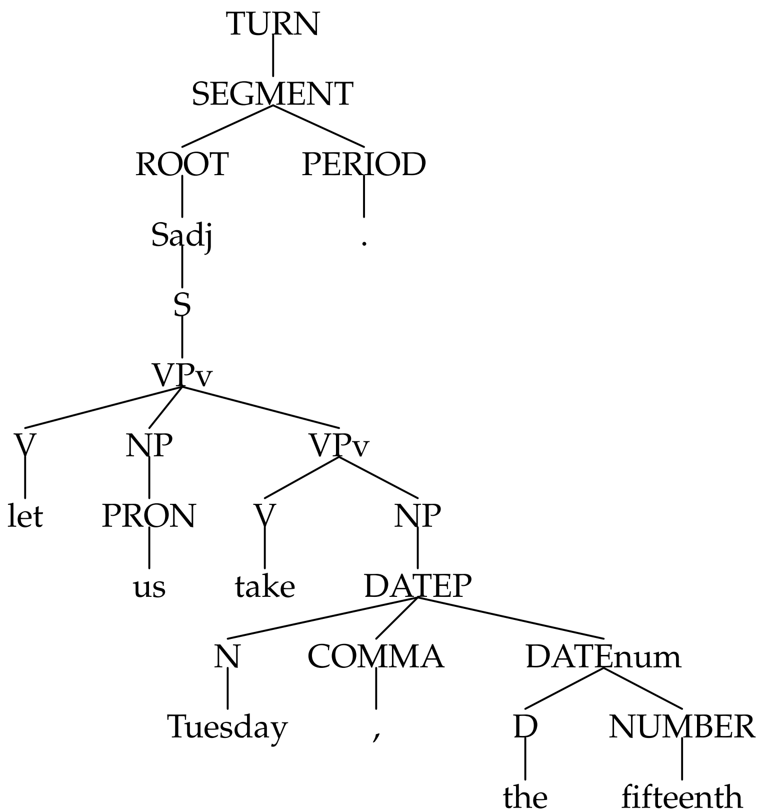
- Syntactic rules

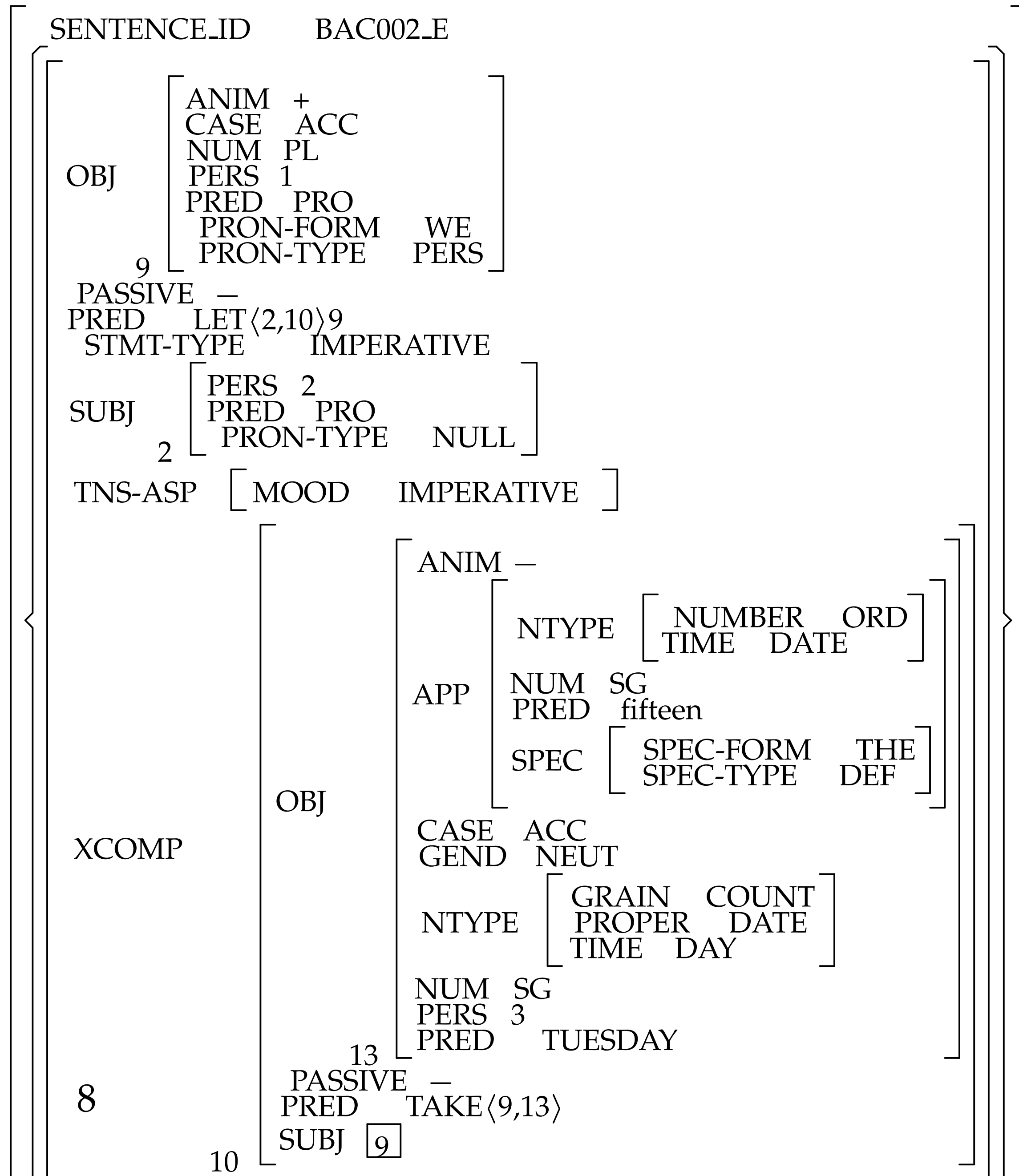
$$S \longrightarrow \underset{\uparrow \mathrm{SUBJ}=\downarrow}{\mathrm{NP}} \; \underset{\uparrow=\downarrow}{\mathrm{VP}} \qquad \mathrm{VP} \longrightarrow \underset{\uparrow=\downarrow}{\mathrm{V}} \; \underset{\uparrow \mathrm{OBJ}=\downarrow}{\mathrm{NP}}$$

- Lexical entries

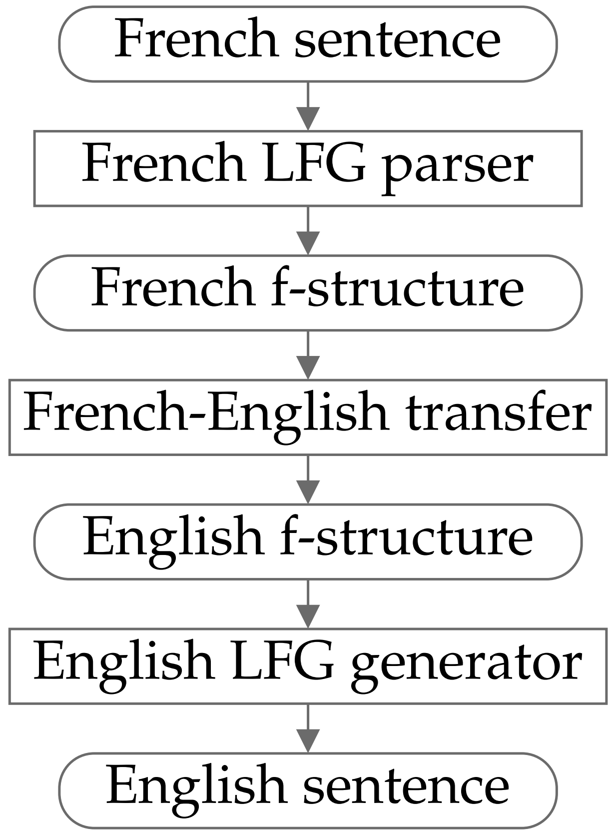
$$\text{Sandy} : \underset{\uparrow \mathrm{PRED}=\mathrm{Sandy}}{\mathrm{NP}} \qquad \text{likes} : \underset{\uparrow \mathrm{PRED}=\mathrm{like}(\uparrow \mathrm{SUBJ\;PRED}, \uparrow \mathrm{OBJ\;PRED})}{\mathrm{V}}$$

Representations of Lexical-Functional Grammar





Machine translation using LFG



- Translation of f-structures seems easier than translation of words
- Ambiguity: each step of the procedure is multi-valued

$\Rightarrow$ If each component is probabilistic, we can identify the most likely translation

Stochastic Lexical-Functional Grammars

- An LFG defines a *set of possible parses* $\mathcal{Y}(x)$ for each sentence x.
- *Features* $f_1, \ldots, f_m$ are real-valued functions on parses
  - Attachment location (high, low, argument, adjunct, etc.)
  - Head-to-head dependencies
- Probability distribution defined by *log-linear model*

$$\begin{aligned} W(y) &= \exp(\sum_{j=1}^{m} \lambda_j f_j(y)) = \prod_{j=1}^{m} \theta_j^{f_j(y)} \\ \Pr(y|w) &= W(y)/Z(w) \end{aligned}$$

where $\theta_j = \exp \lambda_j > 0$ are *feature weights* and
$Z(w) = \sum_{y \in \mathcal{Y}(w)} W(y)$ is the *partition function*.

Johnson et al (1999) "Parsing and estimation for SUBGs", Proc ACL

Features used in our SLFGs

Rule features: For every non-terminal X, $f_X(y)$ is the number of times X occurs in c-structure of y

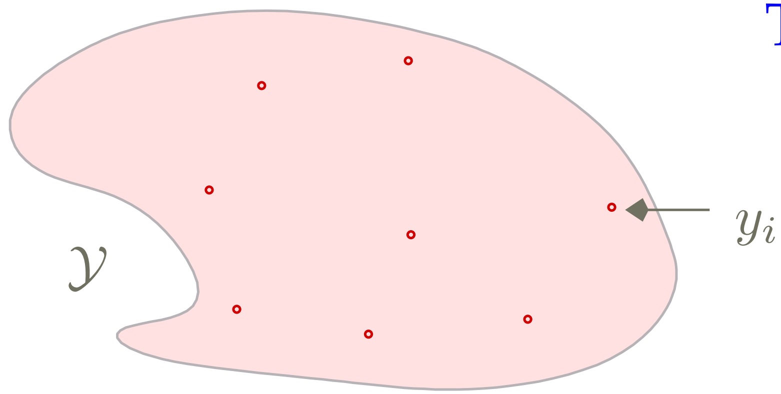
Attribute value features: For every attribute a and every atomic value v, $f_{a=v}(y)$ is the number of times the pair $a = v$ appears in y

Argument and adjunct features: For every grammatical function g, $f_g(\omega)$ is the number of times that g appears in y

Other features: Dates, times, locations; right branching; attachment location; parallelism in coordination; …

Features are *not* independent, but dependency structure is unknown.

ML estimation of feature weights



Training data $D = (y_1, \ldots, y_n)$

$$\hat{\lambda} = \operatorname*{argmax}_{\lambda} L_D(\lambda)$$

$$L_D(\lambda) = \prod_{i=1}^{n} \mathrm{Pr}_\lambda(y_i)$$

$$\mathrm{Pr}_\lambda(y) = \frac{W_\lambda(y)}{Z_\lambda} \quad W_\lambda(y) = \exp \sum_j \lambda_j f_j(y) \quad Z_\lambda = \sum_{y' \in \mathcal{Y}} W_\lambda(y')$$

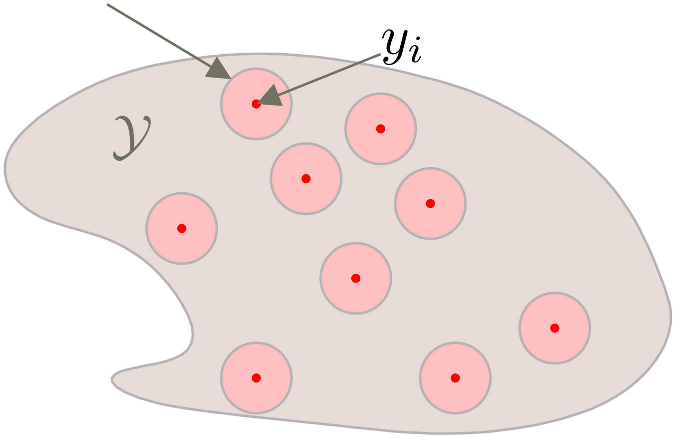
- ML estimation maximizes score $W(y_i)$ of correct parse y_i relative to sum Z of scores of all parses $\mathcal{Y}$
- in general Z_λ and $\partial L_D / \partial \lambda$ are *intractable analytically and numerically* (no effective means to sum over all $\mathcal{Y}$)
- Abney (1997) suggests a Monte-Carlo calculation method

Conditional ML estimation of feature weights

$D = ((x_1, y_1), \ldots, (x_n, y_n))$, where x_i is a string and y_i its parse

Maximize *conditional likelihood* of parses y_i given their strings x_i, i.e., maximize score $W(y_i)$ of correct parse y_i relative to sum $Z(x_i)$ of scores of all parses $\mathcal{Y}(x_i)$ of string x_i

$\mathcal{Y}(x_i) = \{y : X(y) = X(y_i)\}$



$$
\begin{aligned}
\hat{\lambda} &= \operatorname*{argmax}_{\lambda} L_D(\lambda) \\
L_D(\lambda) &= \prod_{i=1}^{n} \Pr_{\lambda}(y_i | x_i) \\
\Pr_{\lambda}(y|x) &= W_{\lambda}(y) / Z_{\lambda}(x)
\end{aligned}
$$

$$
W_{\lambda}(y) = \exp \sum_j \lambda_j f_j(y) \qquad Z_{\lambda}(x) = \sum_{y' \in \mathcal{Y}(x)} W_{\lambda}(y')
$$

Conditional ML versus ML

- The conditional partition function $Z_\lambda(x)$ is *much easier to compute* than the partition function Z_λ
  - Z_λ requires a sum over all parses $\mathcal{Y}$
  - $Z_\lambda(x)$ requires a sum over $\mathcal{Y}(x)$ (parses of string x)
- *Maximum likelihood* estimates full joint distribution
  - learns distribution of both strings and parses given strings
- *Maximum conditional likelihood* estimates a conditional distribution
  - learns distribution of *parses given yields*, but not yields
  - conditional distribution is all you need for parsing
- Conditional estimator is *consistent for the conditional distribution* only

Conditional likelihood estimation

	Correct parse's features	**Features of other parses of same string**
sentence 1	$[1, 3, 2]$	$[2, 2, 3]$ $[3, 1, 5]$ $[2, 6, 3]$
sentence 2	$[7, 2, 1]$	$[2, 5, 5]$
sentence 3	$[2, 4, 2]$	$[1, 1, 7]$ $[7, 2, 1]$
$\ldots$	$\ldots$	$\ldots$

- Training data is *fully observed* (i.e., parsed data)
- Choose λ to maximize (log) likelihood of *correct* parses relative to all parses of same string
- Distribution of *strings* is ignored

Pseudo-constant features are uninformative

	Correct parse's features	Features of other parses of same string
sentence 1	$[1, 3, 2]$	$[2, 2, 2]$ $[3, 1, 2]$ $[2, 6, 2]$
sentence 2	$[7, 2, 5]$	$[2, 5, 5]$
sentence 3	$[2, 4, 4]$	$[1, 1, 4]$ $[7, 2, 4]$
$\ldots$	$\ldots$	$\ldots$

- *Pseudo-constant features* are identical within every set of parses
- They contribute the same constant factor to each parses' likelihood
- They do not distinguish parses of any sentence ⇒irrelevant

Pseudo-maximal features ⇒unbounded $\widehat{\lambda_j}$

	Correct parse's features	Features of other parses of same string
sentence 1	$[1, \mathit{3}, 2]$	$[2, \mathit{3}, 4]$ $[3, \mathit{1}, 1]$ $[2, \mathit{1}, 1]$
sentence 2	$[2, \mathit{7}, 4]$	$[3, \mathit{7}, 2]$
sentence 3	$[2, \mathit{4}, 4]$	$[1, \mathit{1}, 1]$ $[1, \mathit{2}, 4]$

- A *pseudo-maximal feature* always reaches its maximum value within a parse on the correct parse
- If f_j is pseudo-maximal, $\widehat{\lambda_j} \to \infty$ (hard constraint)
- If f_j is pseudo-minimal, $\widehat{\lambda_j} \to -\infty$ (hard constraint)

Regularization helps avoid overlearning

- f_j is pseudo-maximal over training data
  $\not\Rightarrow f_j$ is pseudo-maximal over all strings
- overlearning because of sparse data
- Regularization: Multiply the conditional likelihood by a zero-mean Gaussian with diagonal covariance

$$\hat{\lambda} \;=\; \operatorname*{argmax}_{\lambda} \log L_D(\lambda) - \sum_{j=1}^{m} \frac{\lambda_j^2}{2\sigma_j^2}$$

- Optimize σ_j on heldout data

Optimization of conditional likelihood

- Several algorithms for *maximum conditional likelihood estimation*
  - Various iterative scaling algorithms
  - Conjugate gradient, *L-BFGS* and other numerical optimization algorithms
- These numerical algorithms require *partial derivates* of the conditional likelihood

$$
\begin{aligned}
\frac{\partial \log L_D(\lambda)}{\partial \lambda_j} &= \sum_i f_j(y_i) - \mathrm{E}_\lambda[f_j|x_i] \\
\mathrm{E}_\lambda[f_j|x] &= \sum_{y \in \mathcal{Y}(x)} f_j(y)\,\mathrm{Pr}_\lambda(y|x)
\end{aligned}
$$

Stochastic LFG experiment

- Two parsed LFG corpora provided by Xerox PARC
- Grammars unavailable, but corpus contains all parses and hand-identified correct parse
- Features chosen by inspecting Verbmobil corpus only

	Verbmobil corpus	**Homecentre corpus**
# of sentences	540	980
# of ambiguous sentences	324	424
Av. length of ambig. sentences	13.8	13.1
# of parses	3245	2865
# of features	191	227
# of rule features	59	57

Pseudo-likelihood estimator evaluation

	Verbmobil corpus 324 sentences		**Homecentre corpus** 424 sentences	
	C	$-\log L_D(\lambda)$	C	$-\log L_D(\lambda)$
Baseline estimator	88.8	533.2	136.9	590.7
Conditional ML estimator	180.0	401.3	283.25	580.6

- Test corpus only contains sentences with more than one parse
- C is the number of sentences of the ambiguous held-out test sentences that the model selected correct parses for
- 10-fold cross-validation evaluation
- Combined system performance: 75% of MAP parses are correct

Scaling up: training from partially labeled data

- Coverage of hand-written grammars is poor
  - Move constraints from base LFG to stochastic component
  - ⇒ parses 95% of heldout Penn treebank test corpus
  - ⇒ massive increase in ambiguity
- Lack of labeled training data
  - Can we use the Penn WSJ treebank? (40,000 sentences with hand-constructed parse trees)
  - LFG parses contain much more information than Penn treebank trees
    - The Penn treebank only contains c-structure information, but no f-structure information
    - ⇒ No purely mechanical way of obtaining LFG training trees
    - *Train from partially labelled data*

Partially labelled data

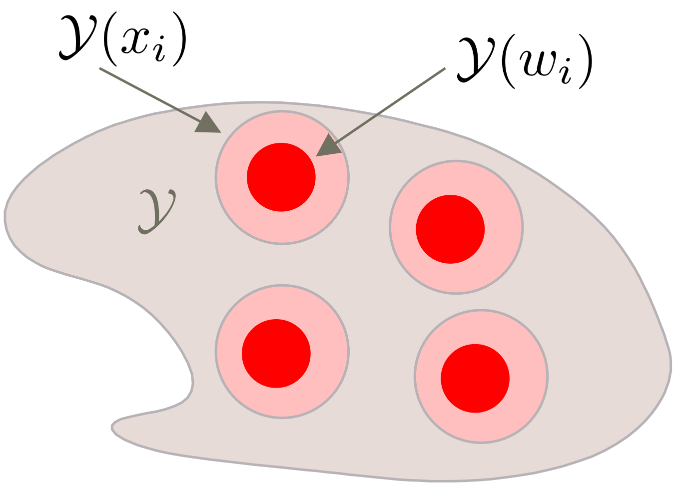
- *Fully labelled training data* identifies the correct parse y_i for each sentence x_i
- *Partially labelled training data* identifies a (small!) set of LFG parses $\mathcal{Y}(w_i)$ which contain y_i
- Obtained mechanically from the WSJ treebank w_i
- $\mathcal{Y}(w)$ the set of LFG parses that:
  - have matching terminal and POS labels
  - have no crossing brackets with the Penn treebank tree w_i
  - agree in the locations of maximal NP, VP and S constituents

Riezler et al (2002) "Parsing the Wall Street Journal using a LFG and Discriminative Estimation Techniques", Proc ACL

Estimation from partially labelled data

$D = ((x_1, w_1), \ldots, (x_n, w_n))$, where x_i is a string and w_i a Penn tree.

$\mathcal{Y}(w_i)$ is the set of LFG parses compatible with w_i; $y_i \in \mathcal{Y}(w_i)$.



$$
\begin{aligned}
L_D(\lambda) &= \prod_{i=1}^{n} \Pr_\lambda(w_i|x_i) \\
\Pr_\lambda(w|x) &= Z_\lambda(w)/Z_\lambda(x) \\
Z_\lambda(x) &= \sum_{y \in \mathcal{Y}(x)} W_\lambda(y) \\
W_\lambda(y) &= \exp \sum_j \lambda_j f_j(y) \\
\frac{\partial L_D(\lambda)}{\partial \lambda_j} &= \sum_{i=1}^{n} \mathrm{E}_\lambda(f_j|w_i) - \mathrm{E}_\lambda(f_j|x_i)
\end{aligned}
$$

Estimation from partially labelled data (cont.)

- Parse each sentence x_i in the training data to obtain set $\mathcal{Y}(x_i)$ of possible parses
- Discard all parses inconsitent with the Penn WSJ treebank parse w_i, producing $\mathcal{Y}(w_i)$
- Numerically optimize a regularized log likelihood based on $\Pr(w|x)$
  - Very similiar to likelihood maximized by EM algorithms

Experiment using Penn WSJ Treebank

- *Discarded unambiguous sentences and sentences longer than 25 words*
- 50% of sentences received a full parse ⇒ 20,000 training sentences
- 500,000 parses from strings alone
- 150,000 parses after treebank filtering
  - Penn WSJ treebank does not add that much information over base LFG grammar
  - Perhaps there are better ways of extracting information from treebank?

Evaluation using Penn WSJ Treebank

- 500 randomly-chosen sentences from section 23 of length ≤ 25 words
- PRED values were extracted *by hand* from parses produced by the grammar
- Evaluated on PRED head-argument matches
- Parser produced 411 full parses and 89 partial parses

	Precision	Recall
Lower bound	75%	79%
Our model	78%	81%
Upper bound	80%	85%

Dynamic programming for parsing and estimation

- As grammar coverage increases, so does ambiguity

  $\Rightarrow$ *How can we improve computational efficiency?*

- Maxwell and Kaplan packed parse representations
- Feature locality (e.g., a f-structure constant)
- Parsing/estimation statistics are sum/max of products
- Graphical representation of product expressions
- Sum/max computations over graphs

Geman and Johnson (2002) “Dynamic programming for parsing and estimation of stochastic unification-based grammars”, Proc ACL

Reparameterization of log linear models

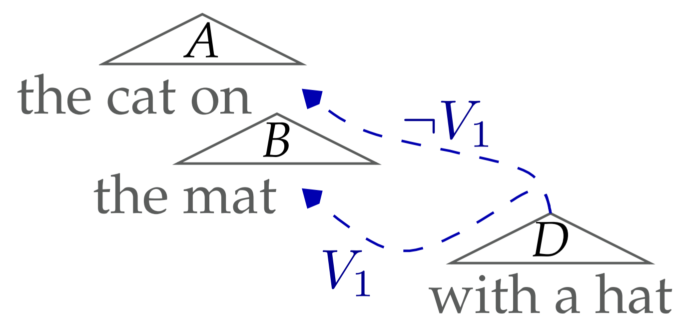
$$\begin{aligned}
\theta_j &= \exp \lambda_j \\
W_\theta(y) &= \exp \sum_{j=1}^{m} \lambda_j f_j(y) \;=\; \prod_{j=1}^{m} \theta_j^{f_j(y)} \\
\Pr_\theta(y|x) &= \frac{W_\theta(y)}{Z_\theta(x)} \\
Z_\theta(x) &= \sum_{y' \in \mathcal{Y}(x)} W_\theta(y')
\end{aligned}$$

- Change of variables permits zero probability events
- $Z_\theta(x)$ involves summing over all possible parses
- Same kind of technique finds most likely parse and calculates $E_\theta[f_j|x]$

Maxwell and Kaplan packed parses

- A parse y consists of set of fragments $\xi \in y$ (MK algorithm)
- A fragment is in a parse when its *context function* is true
- Context functions are functions of *context variables* $V_1, V_2, \ldots$
- The variable assignment must satisfy "no-good" functions
- Each parse is identified by a *unique context variable assignment*

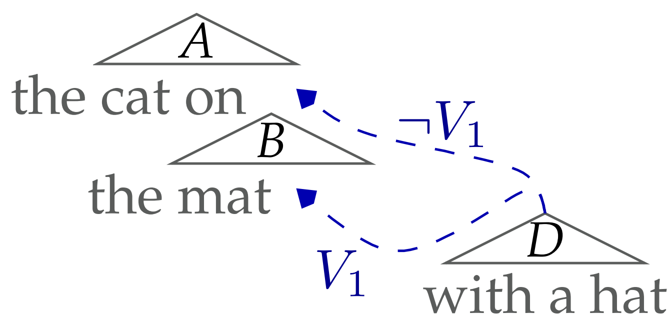
$$\begin{array}{rcl} \xi & = & \textit{"the cat on the mat"} \\ \xi_1 & = & \textit{"with a hat"} \\ V_1 & \to & \text{"attach } D \text{ to } B\text{"} \\ \neg V_1 & \to & \text{"attach } D \text{ to } A\text{"} \end{array}$$



Feature locality

- Features must be *local* to fragments: $f_j(y) = \sum_{\xi \in y} f_j(\xi)$
- May require changes to UBG to make all features local

$$
\begin{array}{rcl}
\xi & = & \text{“\textit{the cat on the mat}”} \\
\xi_1 & = & \text{“\textit{with a hat}”} \\
V_1 & \rightarrow & \text{“attach } D \text{ to } B\text{”} \wedge (\xi_1 \ \text{ATTACH}) = \text{LOW} \\
\neg V_1 & \rightarrow & \text{“attach } D \text{ to } A\text{”} \wedge (\xi_1 \ \text{ATTACH}) = \text{HIGH}
\end{array}
$$



Feature locality decomposes $W(y)$

- Feature locality: the weight of a parse is the product of weights of its fragments

$$
\begin{aligned}
W(y) &= \prod_{\xi \in y} W(\xi), \qquad \text{where} \\
W(\xi) &= \prod_{j=1}^{m} \theta_j^{f_j(\xi)}
\end{aligned}
$$

$$
\begin{aligned}
&W(\xi = \textit{“the cat on the mat”}) \\
&W(\xi_1 = \textit{“with a hat”}) \\
V_1 \quad \rightarrow \quad &W\,(\text{“attach } D \text{ to } B\text{”} \;\wedge\; (\xi_1 \text{ ATTACH}) = \text{LOW}\,) \\
\neg V_1 \quad \rightarrow \quad &W\,(\text{“attach } D \text{ to } A\text{”} \;\wedge\; (\xi_1 \text{ ATTACH}) = \text{HIGH}\,)
\end{aligned}
$$

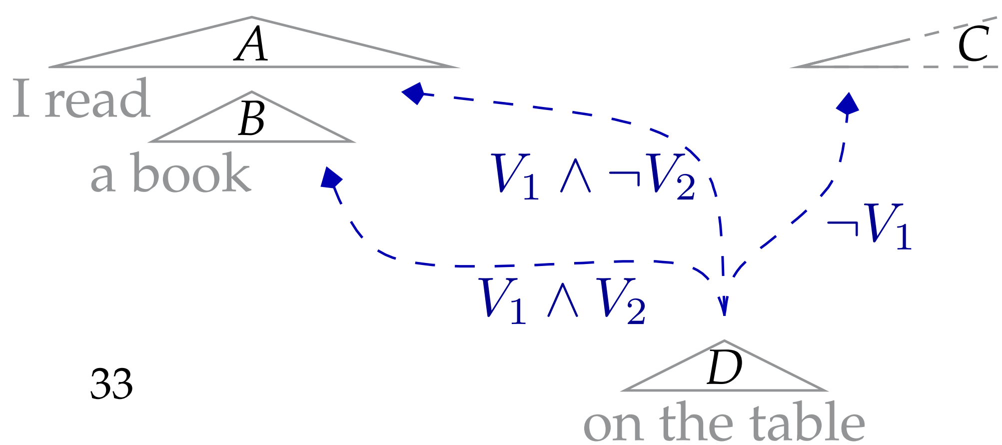
No-goods and impossible variable assignments

- Not all variable assignments correspond to parses
- *A no-good is a function $\eta(v)$ that is false when v doesn't correspond to a parse*
- $\eta(v) = 0 \rightarrow v$ has zero probability

$$\begin{aligned} \xi &= \text{“\textit{I read a book}”} \\ \xi_1 &= \text{“\textit{on the table}”} \end{aligned}$$

$$\begin{aligned} V_1 \wedge V_2 &\rightarrow \text{“attach } D \text{ to } B\text{”} \\ V_1 \wedge \neg V_2 &\rightarrow \text{“attach } D \text{ to } A\text{”} \\ \neg V_1 &\rightarrow \text{“attach } D \text{ to } C\text{”} \end{aligned}$$

$V_1 \vee V_2$



Identify parses with variable assignments

- For a given sentence x, *a variable assignment v to V uniquely identifies a parse y*
- Let $W(v) = W(y)$ where y is the parse identified by v

$\Rightarrow$ Argmax/sum/expectations over parses can be computed over context variables V instead of over complete parses

Most likely parse: $\hat{x} = \operatorname*{argmax}_{v \in \mathcal{V}(x)} W(v)$

Partition function: $Z(x) = \sum_{v \in \mathcal{V}(x)} W(v)$

Expectation:* $E[f_j|x] = \sum_{v \in \mathcal{V}(x)} f_j(v) W(v) / Z(x)$

$W(v)$ is a product of functions of v

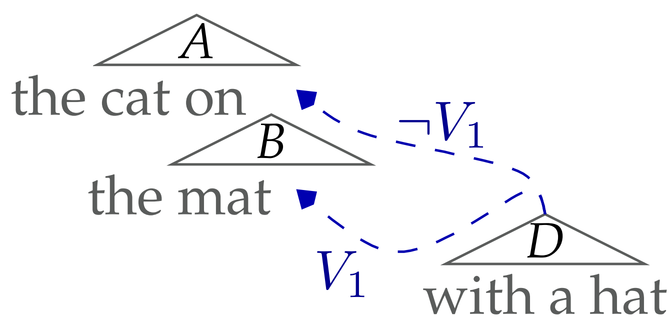
- $W(v) = \prod_{A \in \mathcal{A}} A(v)$, where:
  - Each line $\alpha(v) \to \xi$ introduces a term $W(\xi)^{\alpha(v)}$
  - Each "no-good" $\eta(v)$ introduces a term $\eta(v)$ (which is zero on variable assignments that do not correspond to parses)

$$
\begin{array}{ccclcc}
 & \vdots & & & & \vdots \\
\alpha(v) & \to & \xi & & \times & W(\xi)^{\alpha(v)} \\
 & \vdots & & & \times & \vdots \\
 & & \eta(v) & & \times & \eta(v) \\
 & \vdots & & & \times & \vdots
\end{array}
$$

$\Rightarrow$ W is a *Markov Random Field* over the context variables V

W is a product of functions of V

$$
\begin{aligned}
W'(V_1) \;=\; & \quad W(\xi \;=\; \text{“\textit{the cat on the mat}”}) \\
\times \; & \quad W(\xi_1 \;=\; \text{“\textit{with a hat}”}) \\
\times \; & \quad W\left(\text{“attach } D \text{ to } B\text{”} \;\wedge\; (\xi_1 \;\textsc{attach}) = \textsc{low}\;\right)^{V_1} \\
\times \; & \quad W\left(\text{“attach } D \text{ to } A\text{”} \;\wedge\; (\xi_1 \;\textsc{attach}) = \textsc{high}\;\right)^{\neg V_1}
\end{aligned}
$$



Product expressions and graphical models

- MRFs are products of terms, each of which is a function of (a few) variables
- Graphical models provide *dynamic programming algorithms* for Markov Random Fields (MRF) (Pearl 1988)
- These algorithms implicitly *factorize the product*
- They generalize the Viterbi and Forward-Backward algorithms to arbitrary graphs (Smyth 1997)

$\Rightarrow$ Graphical models provide dynamic programming techniques for parsing and training Stochastic UBGs

Factorization example

$$\begin{aligned} W'(V_1) &= & & W(\xi = \textit{“the cat on the mat”}) \\ & & \times\ & W(\xi_1 = \textit{“with a hat”}) \\ & & \times\ & W\left(\text{“attach } D \text{ to } B\text{”} \wedge (\xi_1\ \textsc{attach}) = \textsc{low}\right)^{V_1} \\ & & \times\ & W\left(\text{“attach } D \text{ to } A\text{”} \wedge (\xi_1\ \textsc{attach}) = \textsc{high}\right)^{\neg V_1} \end{aligned}$$

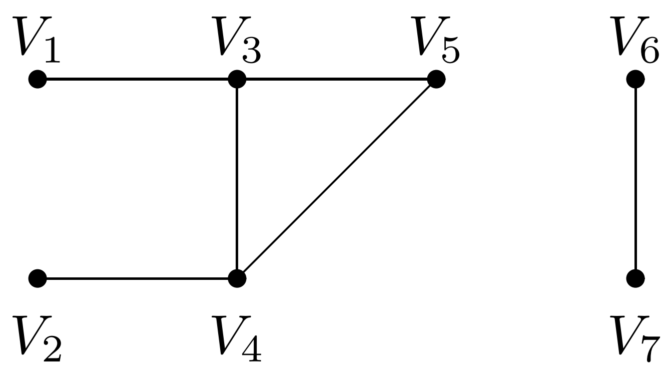
$$\begin{aligned} \max_{V_1} W'(V_1) &= & & W(\xi = \textit{“the cat on the mat”}) \\ & & \times\ & W(\xi_1 = \textit{“with a hat”}) \\ & & \times\ & \max_{V_1} \begin{pmatrix} W\left(\text{“attach } D \text{ to } B\text{”} \wedge (\xi_1\ \textsc{attach}) = \textsc{low}\right)^{V_1}, \\ W\left(\text{“attach } D \text{ to } A\text{”} \wedge (\xi_1\ \textsc{attach}) = \textsc{high}\right)^{\neg V_1} \end{pmatrix} \end{aligned}$$

Dependency structure graph $G_{\mathcal{A}}$

$$Z(x) = \sum_v W(v) = \sum_v \prod_{A \in \mathcal{A}} A(v)$$

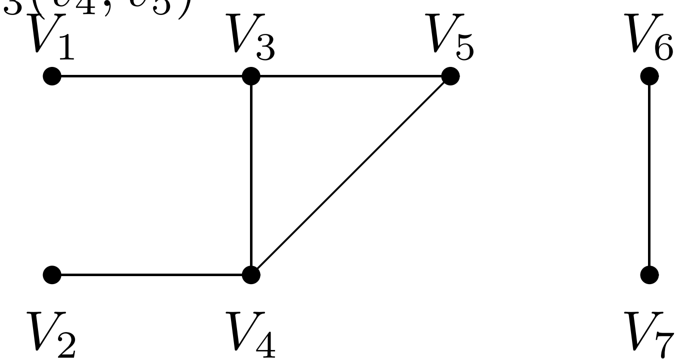
- $G_{\mathcal{A}}$ is the *dependency graph* for $\mathcal{A}$
  - context variables X are vertices of $G_{\mathcal{A}}$
  - $G_{\mathcal{A}}$ has an edge (v_i, v_j) if both are arguments of some $A \in \mathcal{A}$

    $A(V) = a(V_1, V_3)b(V_2, V_4)c(V_3, V_4, V_5)d(V_4, V_5)e(V_6, V_7)$



Graphical model computations

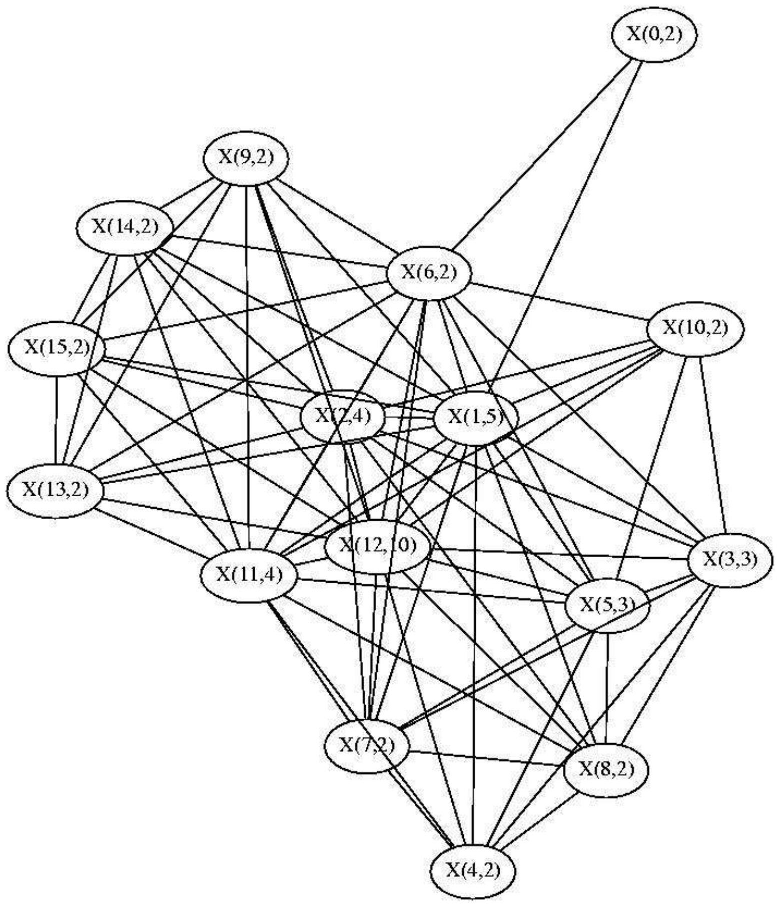
$$
\begin{aligned}
Z &= \sum_v a(v_1, v_3) b(v_2, v_4) c(v_3, v_4, v_5) d(v_4, v_5) e(v_6, v_7) \\
Z_1(v_3) &= \sum_{v_1} a(v_1, v_3) \\
Z_2(v_4) &= \sum_{v_2} b(v_2, v_4) \\
Z_3(v_4, v_5) &= \sum_{v_3} c(v_3, v_4, v_5) Z_1(v_3) \\
Z_4(v_5) &= \sum_{v_4} d(v_4, v_5) Z_2(v_4) Z_3(v_4, v_5) \\
Z_5 &= \sum_{v_5} Z_4(v_5) \\
Z_6(v_7) &= \sum_{v_6} e(v_6, v_7) \\
Z_7 &= \sum_{v_7} Z_6(v_7) \\
Z &= Z_5 Z_7 \\
&= \left(\sum_{v_5} Z_4(v_5)\right) \left(\sum_{v_7} Z_6(v_7)\right)
\end{aligned}
$$



See: Pearl (1988) *Probabilistic Reasoning in Intelligent Systems*

Graphical model for Homecentre example

Use a damp, lint-free cloth to wipe the dust and dirt buildup from the scanner plastic window and rollers.



Computational complexity

- Polynomial in m = the *maximum number of variables in the dynamic programming functions* $\geq$ the number of variables in any function A
- m depends on the ordering of variables (and G)
- Finding the variable ordering that minimizes m is NP-complete, but there are good heuristics

$\Rightarrow$ Worst case exponential (no better than enumerating the parses), but average case might be much better

  - Much like UBG parsing complexity

Summary

- Parse selection can be formulated as a classification problem

  $\Rightarrow$ Virtually any classification algorithm can be used

- The training data often only partially identifies the correct parse

  $\Rightarrow$ Classification algorithms that can train from *partially labeled data*

- Combinatorial explosion in number of parses

  $\Rightarrow$ Reformulate parsing and estimation problems as *MRF graphical model problems*

Future directions

- Reformulate “hard” grammatical constraints as “soft” stochastic features
  - Underlying grammar permits all possible structural combinations
  - Grammatical constraints reformulated as stochastic features

  $\Rightarrow$ computationally efficiency will be even more important
- Better methods for learning from partially labeled data (e.g., co-training)
- Feature selection, automatic feature induction
- Applications such as *machine translation* and *automatic summarization*