Statistical Parsing

Christopher Manning
CS224N

Example of uniform cost search vs. CKY parsing: The grammar, lexicon, and sentence

- S → NP VP %% 0.9
- S → VP %% 0.1
- VP → V NP %% 0.6
- VP → V %% 0.4
- NP → NP NP %% 0.3
- NP → N %% 0.7

- N → people %% 0.8
- N → fish %% 0.1
- N → tanks %% 0.1
- V → people %% 0.1
- V → fish %% 0.6
- V → tanks %% 0.3

- *people fish tanks*

Example of uniform cost search vs. CKY parsing: CKY vs. order of agenda pops in chart

```
N[0,1] -> people  %% 0.8              %% [0,1]
V[0,1] -> people  %% 0.1
NP[0,1] -> N[0,1]  %% 0.56
VP[0,1] -> V[0,1]  %% 0.04
S[0,1] -> VP[0,1]  %% 0.004
N[1,2] -> fish  %% 0.1                %% [1,2]
V[1,2] -> fish  %% 0.6
NP[1,2] -> N[1,2]  %% 0.07
VP[1,2] -> V[1,2]  %% 0.24
S[1,2] -> VP[1,2]  %% 0.024
N[2,3] -> tanks  %% 0.1               %% [2,3]
V[2,3] -> fish  %% 0.3
NP[2,3] -> N[2,3]  %% 0.07
VP[2,3] -> V[2,3]  %% 0.12
S[2,3] -> VP[2,3]  %% 0.012
NP[0,2] -> NP[0,1] NP[1,2]  %% 0.01176  %% [0,2]
VP[0,2] -> V[0,1] NP[1,2]  %% 0.0042
S[0,2] -> NP[0,1] VP[1,2]  %% 0.12096
S[0,2] -> VP[0,2]  %% 0.00042
NP[1,3] -> NP[1,2] NP[2,3]  %% 0.00147  %% [1,3]
VP[1,3] -> V[1,2] NP[2,3]  %% 0.0252
S[1,3] -> NP[1,2] VP[2,3]  %% 0.00756
S[1,3] -> VP[1,3]  %% 0.00252
S[0,3] -> NP[0,1] VP[1,3]  %% 0.0127008 %% [0,3] Best
S[0,3] -> NP[0,2] VP[2,3]  %% 0.0021168
VP[0,3] -> V[0,1] NP[1,3]  %% 0.0000882
NP[0,3] -> NP[0,1] NP[1,3]  %% 0.00024696
NP[0,3] -> NP[0,2] NP[2,3]  %% 0.00024696
S[0,3] -> VP[0,3]  %% 0.00000882
```

```
N[0,1] -> people  %% 0.8
V[1,2] -> fish  %% 0.6
NP[0,1] -> N[0,1]  %% 0.56
V[2,3] -> fish  %% 0.3
VP[1,2] -> V[1,2]  %% 0.24
S[0,2] -> NP[0,1] VP[1,2]  %% 0.12096
VP[2,3] -> V[2,3]  %% 0.12
V[0,1] -> people  %% 0.1
N[1,2] -> fish  %% 0.1
N[2,3] -> tanks  %% 0.1
NP[1,2] -> N[1,2]  %% 0.07
NP[2,3] -> N[2,3]  %% 0.07
VP[0,1] -> V[0,1]  %% 0.04
VP[1,3] -> V[1,2] NP[2,3]  %% 0.0252
S[1,2] -> VP[1,2]  %% 0.024
S[0,3] -> NP[0,1] VP[1,3]  %% 0.0127008          Best
----
S[2,3] -> VP[2,3]  %% 0.012
NP[0,2] -> NP[0,1] NP[1,2]  %% 0.01176
S[1,3] -> NP[1,2] VP[2,3]  %% 0.00756
VP[0,2] -> V[0,1] NP[1,2]  %% 0.0042
S[0,1] -> VP[0,1]  %% 0.004
S[1,3] -> VP[1,3]  %% 0.00252
NP[1,3] -> NP[1,2] NP[2,3]  %% 0.00147
NP[0,3] -> NP[0,2] NP[2,3]  %% 0.00024696
```

What can go wrong in parsing?

- We can build too many items.
  - Most items that can be built, shouldn't.
  - CKY builds them all!

Speed: build promising items first.

- We can build in an bad order.
  - Might find bad parses for parse item before good parses.
  - Will trigger best-first propagation.

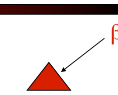
Correctness: keep items on the agenda until you're sure you've seen their best parse.

Speeding up agenda-based parsers

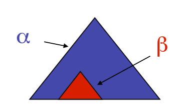
- Two options for doing less work
  - The optimal way: A* parsing
    - Klein and Manning (2003)
  - The ugly but much more practical way: "best-first" parsing
    - Caraballo and Charniak (1998)
    - Charniak, Johnson, and Goldwater (1998)

A* Search

- Problem with uniform-cost:
  - Even unlikely small edges have high score.
  - We end up processing every small edge!
- Solution: A* Search
  - Small edges have to fit into a full parse.
  - The smaller the edge, the more the full parse will cost [cost = (neg. log prob)].
  - Consider both the cost to build (β) and the cost to complete (α).
- We figure out β during parsing.
- We GUESS at α in advance (pre-processing).
  - Exactly calculating this quantity is as hard as parsing.
  - But we can do A* parsing if we can cheaply calculate *underestimates* of the true cost



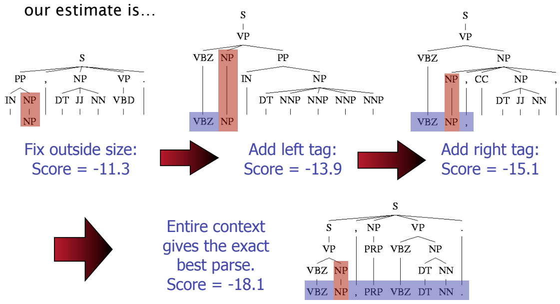
Score = β



Score = β + α

Using context for admissable outside estimates

- The more detailed the context used to estimate α is, the sharper our estimate is…

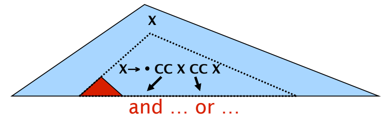


Categorical filters are a limit case of A* estimates

- Let projection π collapse all phrasal symbols to "X":

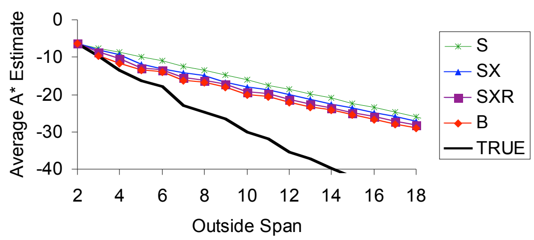
NP → X
NP→ • CC NP CC NP → X→ • CC X CC X (π)

- When can X→ • CC X CC X be completed?



- Whenever the right context includes two CCs!
- Gives an admissible lower bound for this projection that is very efficient to calculate.

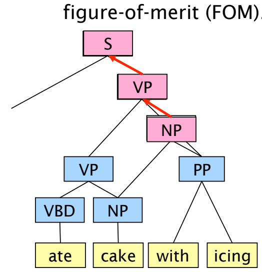
A* Context Summary Sharpness



Adding local information changes the intercept, but not the slope!

Best-First Parsing

- In best-first, parsing, we visit edges according a figure-of-merit (FOM).



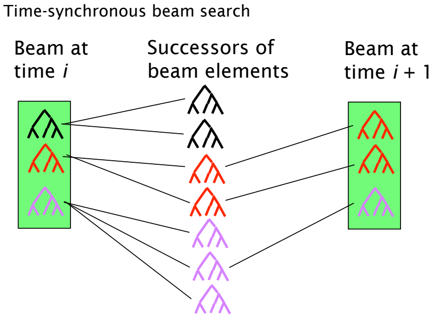
- A good FOM focuses work on "quality" edges.
- The good: leads to full parses quickly.
- The (potential) bad: leads to non-MAP parses.
- The ugly: propagation
  - If we find a better way to build a parse item, we need to rebuild everything above it
- In practice, works well!

Beam Search

- State space search
- States are partial parses with an associated probability
  - Keep only the top scoring elements at each stage of the beam search
- Find a way to ensure that all parses of a sentence have the same number *N* steps
  - Or at least are roughly comparable
  - Leftmost top-down CFG derivations in *true* CNF
  - Shift-reduce derivations in *true* CNF
  - Partial parses that cover the same number of words

Beam Search

- Time-synchronous beam search



Kinds of beam search

- Constant beam size *k*
- Constant beam width relative to best item
  - Defined either additively or multiplicatively
- Sometimes combination of the above two
- Sometimes do fancier stuff like trying to keep the beam elements diverse
- Beam search can be made very fast
- No measure of how often you find model optimal answer
  - But can track correct answer to see how often/far gold standard optimal answer remains in the beam

Beam search treebank parsers?

- Most people do bottom up parsing (CKY, shift-reduce parsing or a version of left-corner parsing)
  - For treebank grammars, not much grammar constraint, so want to use data-driven constraint
  - Adwait Ratnaparkhi 1996 [maxent shift-reduce parser]
  - Manning and Carpenter 1998 and Henderson 2004 left-corner parsers
- But top-down with rich conditioning is possible
  - Cf. Brian Roark 2001
- Don't actually want to store states as partial parses
  - Store them as the last rule applied, with backpointers to the previous states that built those constituents (and a probability)
  - You get a linear time parser … but you may not find the best parses according to your model (things "fall off the beam")

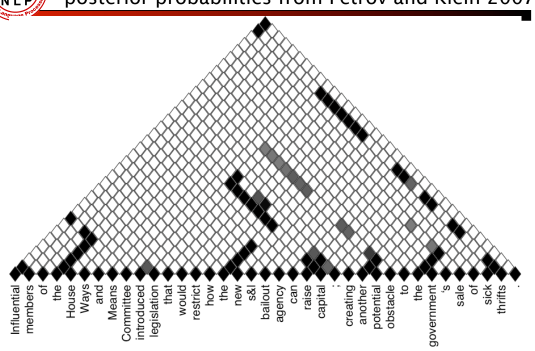
Search in modern lexicalized statistical parsers

- Klein and Manning (2003b) do optimal A* search
  - Done in a restricted space of lexicalized PCFGs that "factors", allowing very efficient A* search
- Collins (1999) exploits both the ideas of beams and agenda based parsing
  - He places a separate beam over each span, and then, roughly, does uniform cost search
- Charniak (2000) uses inadmissible heuristics to guide search
  - He uses very good (but inadmissible) heuristics – "best first search" – to find good parses quickly
  - Perhaps unsurprisingly this is the fastest of the 3.

Coarse-to-fine parsing

- Uses grammar projections to guide search
  - VP-VBF, VP-VBG, VP-U-VBN, … → VP
  - VP[*buys*/VBZ], VP[*drive*/VB], VP[*drive*/VBP], … → VP
  - You can parse much more quickly with a simple grammar because the grammar constant is way smaller
  - You restrict the search of the expensive refined model to explore only spans and/or spans with compatible labels that the simple grammar liked
- Very successfully used in several recent parsers
  - Charniak and Johnson (2005)
  - Petrov and Klein (2007)

Coarse-to-fine parsing: A visualization of the span posterior probabilities from Petrov and Klein 2007

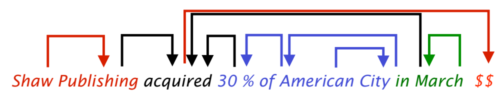


Dependency parsing

Dependency Grammar/Parsing

- A sentence is parsed by relating each word to other words in the sentence which depend on it.
- The idea of dependency structure goes back a long way
  - To Pāṇini's grammar (c. 5th century BCE)
- Constituency is a new-fangled invention
  - 20th century invention (R.S. Wells, 1947)
- Modern dependency work often linked to work of L. Tesnière (1959)
  - Dominant approach in "East" (Russia, China, …)
  - Basic approach of 1st millennium Arabic grammarians
- Among the earliest kinds of parsers in NLP, even in the US:
  - David Hays, one of the founders of computational linguistics, built early (first?) dependency parser (Hays 1962)

Dependency structure

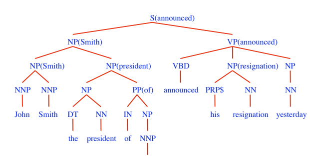


- Words are linked from head (regent) to dependent
- Warning! Some people do the arrows one way; some the other way (Tesniere has them point from head to dependent…)
- Usually add a fake ROOT (here $$) so every word is a dependent of precisely 1 other node

Relation between CFG to dependency parse

- A dependency grammar has a notion of a head
- Officially, CFGs don't
- But modern linguistic theory and all modern statistical parsers (Charniak, Collins, Stanford, …) do, via hand-written phrasal "head rules":
  - The head of a Noun Phrase is a noun/number/adj/…
  - The head of a Verb Phrase is a verb/modal/….
- The head rules can be used to extract a dependency parse from a CFG parse (follow the heads).
- A phrase structure tree can be got from a dependency tree, but dependents are flat (no VP!)

Propagating head words

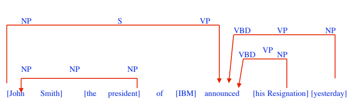


- Small set of rules propagate heads

Extracted structure

NB. Not all dependencies shown here

- Dependencies are inherently untyped, though some work like Collins (1996) types them using the phrasal categories



Quiz question!

- Draw a dependency diagram (with arrows pointing from dependent to head) for:

Retail sales drop in April cools afternoon market trading

Dependency Conditioning Preferences

Sources of information:

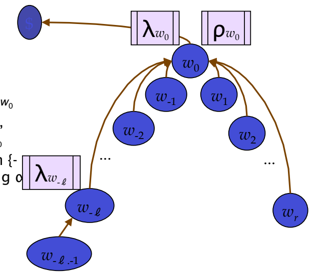
- bilexical dependencies
- distance of dependencies
- valency of heads (number of dependents)

A word's **dependents** (adjuncts, arguments) tend to fall **near** it in the string.

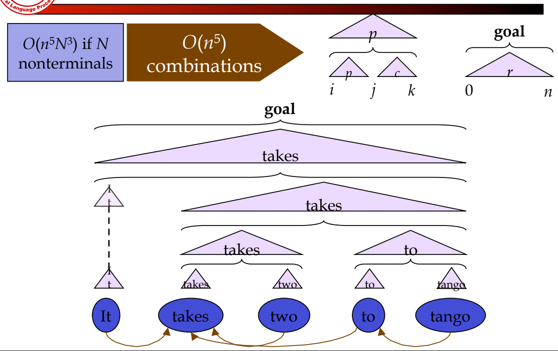
These next 6 slides are based on slides by Jason Eisner and Noah Smith

Probabilistic dependency grammar: generative model

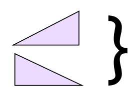
1. Start with left wall \$
2. Generate root w_0
3. Generate left children w_{-1}, w_{-2}, ..., $w_{-\ell}$ from the FSA λ_{w_0}
4. Generate right children w_1, w_2, ..., w_r from the FSA ρ_{w_0}
5. Recurse on each w_i for i in {-ℓ, ..., -1, 1, ..., r}, sampling α_i (steps 2-4)
6. Return $\alpha_{\ell}...\alpha_{-1}w_0\alpha_1...\alpha_r$



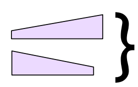
Naïve Recognition/Parsing



Dependency Grammar Cubic Recognition/Parsing (Eisner & Satta, 1999)

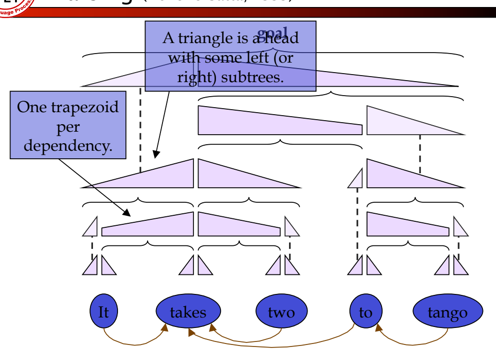


- ***Triangles***: span over words, where tall side of triangle is the head, other side is dependent, and no non-head words expecting more dependents

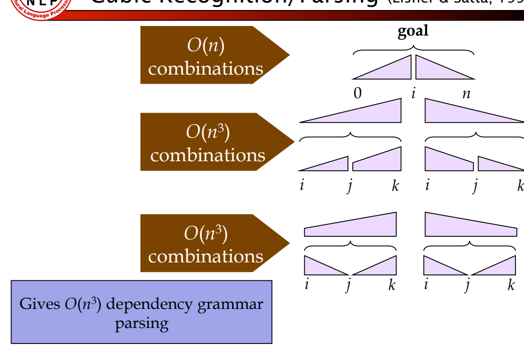


- ***Trapezoids***: span over words, where larger side is head, smaller side is dependent, and smaller side is still looking for dependents on its side of the trapezoid

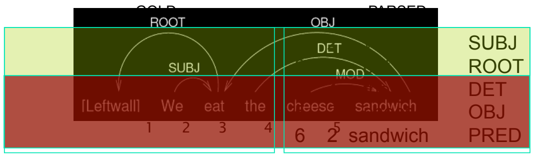
Dependency Grammar Cubic Recognition/Parsing (Eisner & Satta, 1999)



Cubic Recognition/Parsing (Eisner & Satta, 1999)



Evaluation of Dependency Parsing: Simply use (labeled) dependency accuracy



Accuracy = number of correct dependencies / total number of dependencies = 2 / 5 = 0.40 40%

1	2	We	SUBJ
2	0	eat	ROOT
4	5	the	DET
5	2	cheese	MOD
6	2	sandwich	SUBJ

McDonald et al. (2005 ACL): Online Large-Margin Training of Dependency Parsers

- Builds a discriminative dependency parser
- Can condition on rich features in that context
  - Best-known recent dependency parser
  - Lots of recent dependency parsing activity connected with CoNLL 2006/2007 shared task
- Doesn't/can't report constituent LP/LR, but evaluating dependencies correct:
  - Accuracy is similar to but a fraction below dependencies extracted from Collins:
    - 90.9% vs. 91.4% … combining them gives 92.2% [all lengths]
    - Stanford parser on length up to 40:
      - Pure generative dependency model: 85.0%
      - Lexicalized factored parser: 91.0%

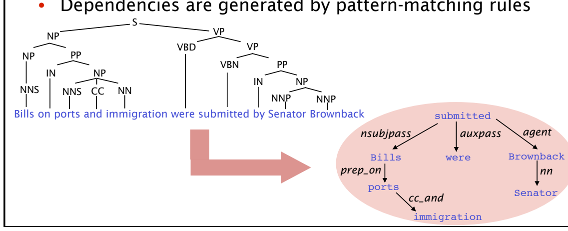
McDonald et al. (2005 ACL): Online Large-Margin Training of Dependency Parsers

- Score of a parse is the sum of the scores of its dependencies
- Each dependency is a linear function of features times weights
- Feature weights are learned by MIRA, an online large-margin algorithm
  - But you could think of it as using a perceptron or maxent classifier
- Features cover:
  - Head and dependent word and POS separately
  - Head and dependent word and POS bigram features
  - Words between head and dependent
  - Length and direction of dependency

Extracting grammatical relations from statistical constituency parsers

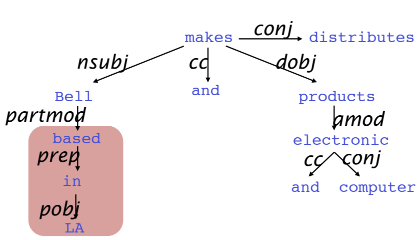
[de Marneffe et al. LREC 2006]

- Exploit the high-quality syntactic analysis done by statistical constituency parsers to get the grammatical relations [typed dependencies]
- Dependencies are generated by pattern-matching rules

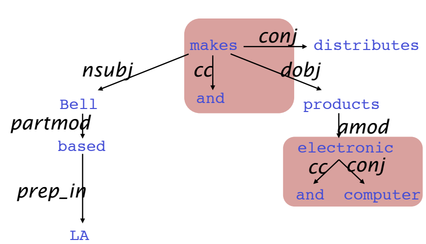


Collapsing to facilitate semantic analysis

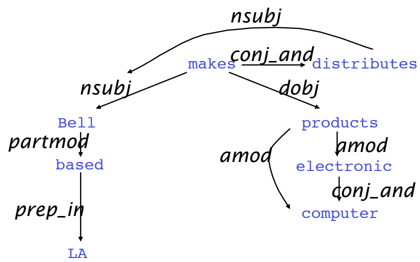
Bell, based in LA, makes and distributes electronic and computer products.



Collapsing to facilitate semantic analysis

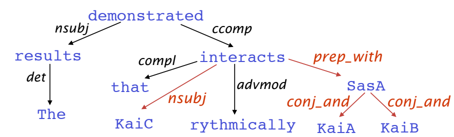


Collapsing to facilitate semantic analysis



Dependency paths to identify IE relations like protein interaction

[Erkan et al. EMNLP 07, Fundel et al. 2007]



KaiC - nsubj - interacts - prep_with - SasA
KaiC - nsubj - interacts - prep_with - SasA - conj_and - KaiA
KaiC - nsubj - interacts - prep_with - SasA - conj_and - KaiB

SasA - conj_and - KaiA
SasA - conj_and - KaiB
KaiA - conj_and - SasA - conj_and - KaiB

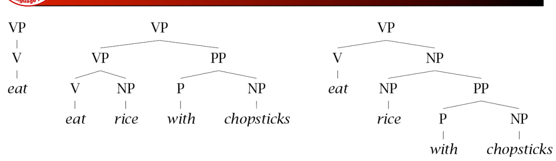
Discriminative Parsing

Discriminative Parsing as a classification problem

- Classification problem
  - Given a training set of iid samples T={(X_1,Y_1) … (X_n,Y_n)} of input and class variables from an unknown distribution D(X,Y), estimate a function $\hat{h}(X)$ that predicts the class from the input variables
- The observed X's are the sentences.
- The class Y of a sentence is its parse tree
- The model has a large (infinite!) space of classes, but we can still assign them probabilities
  - The way we can do this is by breaking whole parse trees into component parts

1. Distribution-free methods
2. Probabilistic model methods

Motivating discriminative estimation (1)



100 6 2

A training corpus of 108 (imperative) sentences.

Based on an example by Mark Johnson

Motivating discriminative estimation (2)

- In discriminative models, it is easy to incorporate different kinds of features
  - Often just about anything that seems linguistically interesting
- In generative models, it's often difficult, and the model suffers because of false independence assumptions
- This ability to add informative features is the real power of discriminative models for NLP.
  - Can still do it for parsing, though it's trickier.

Discriminative Parsers

- Discriminative Dependency Parsing
  - Not as computationally hard (tiny grammar constant)
  - Explored considerably recently. E.g. McDonald et al. 2005
- Make parser action decisions discriminatively
  - E.g. with a shift-reduce parser
- Dynamic-programmed Phrase Structure Parsing
  - Resource intensive! Most work on sentences of length <=15
  - The need to be able to dynamic program limits the feature types you can use
- Post-Processing: Parse reranking
  - Just work with output of k-best generative parser

Discriminative models

Shift-reduce parser Ratnaparkhi (1998)

- Learns a distribution P(T|S) of parse trees given sentences using the sequence of actions of a shift-reduce parser

$$P(T \mid S) = \prod_{i=1}^{n} P(a_i \mid a_1 \ldots a_{i-1} S)$$

- Uses a maximum entropy model to learn conditional distribution of parse action given history
- Suffers from independence assumptions that actions are independent of future observations as with CMM/MEMM
- Higher parameter estimation cost to learn local maximum entropy models
- **Lower** but still good accuracy: 86% - 87% labeled precision/recall

Discriminative dynamic-programmed parsers

- Taskar et al. (2004 EMNLP) show how to do joint discriminative SVM-style ("max margin) parsing building a phrase structure tree also conditioned on words in O(n^3) time
  - In practice, totally impractically slow. Results were never demonstrated on sentences longer than 15 words
- Turian et al. (2006 NIPS) do a decision-tree based discriminative parser
- Finkel, et al. (2008 ACL) feature-based discriminative parser is just about practical.
  - Does dynamic programming discriminative parsing of long sentences (train and test on up to 40 word sentences)
  - 89.0 LP/LR F1

Discriminative Models – Distribution Free Re-ranking (Collins 2000)

- Represent sentence-parse tree pairs by a feature vector F(X,Y)
- Learn a linear ranking model with parameters $\overline{\alpha}$ using the boosting loss

Model	LP	LR
Collins 99 (Generative)	88.3%	88.1%
Collins 00 (BoostLoss)	89.9%	89.6%

13% error reduction

Still very close in accuracy to generative model [Charniak 2000]

Charniak and Johnson (2005 ACL):
Coarse-to-fine *n*-best parsing and MaxEnt discriminative reranking

- Builds a maxent discriminative reranker over parses produced by (a slightly bugfixed and improved version of) Charniak (2000).
- Gets 50 best parses from Charniak (2000) parser
  - Doing this exploits the "coarse-to-fine" idea to heuristically find good candidates
- Maxent model for reranking uses heads, etc. as generative model, but also nice linguistic features:
  - Conjunct parallelism
  - Right branching preference
  - Heaviness (length) of constituents factored in
- Gets 91% LP/LR F1 (on *all* sentences! – up to 80 wd)