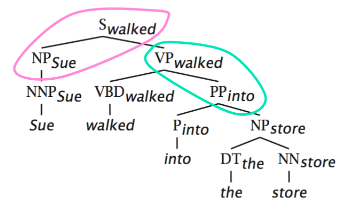
Statistical Parsing

Christopher Manning
CS224N

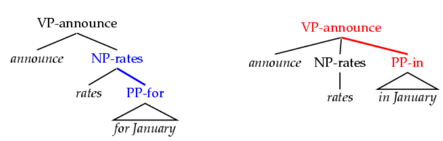
(Head) Lexicalization of PCFGs
[Magerman 1995, Collins 1997; Charniak 1997]

- The head word of a phrase gives a good representation of the phrase's structure and meaning
- Puts the properties of words back into a PCFG



(Head) Lexicalization of PCFGs
[Magerman 1995, Collins 1997; Charniak 1997]

- Word-to-word affinities are useful for certain ambiguities
  - See how PP attachment is (partly) captured in a local PCFG rule. What isn't captured?



Lexicalized Parsing was seen as the breakthrough of the late 90s

- Eugene Charniak, 2000 JHU workshop: "To do better, it is necessary to condition probabilities on the actual words of the sentence. This makes the probabilities much tighter:
  - $p(\text{VP} \rightarrow \text{V NP NP})$ = 0.00151
  - $p(\text{VP} \rightarrow \text{V NP NP} \mid \text{said})$ = 0.00001
  - $p(\text{VP} \rightarrow \text{V NP NP} \mid \text{gave})$ = 0.01980 "
- Michael Collins, 2003 COLT tutorial: "Lexicalized Probabilistic Context-Free Grammars … perform vastly better than PCFGs (88% vs. 73% accuracy)"

Michael Collins (2003, COLT)

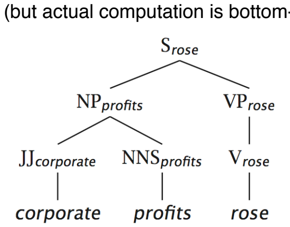
Results

F_1 (!)

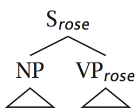
Method	Accuracy
PCFGs (Charniak 97)	73.0%
Conditional Models – Decision Trees (Magerman 95)	84.2%
Lexical Dependencies (Collins 96)	85.5%
Conditional Models – Logistic (Ratnaparkhi 97)	86.9%
Generative Lexicalized Model (Charniak 97)	86.7%
Generative Lexicalized Model (Collins 97)	88.2%
Logistic-inspired Model (Charniak 99)	89.6%
Boosting (Collins 2000)	89.8%

Parsing via classification decisions: Charniak (1997)

- A very simple, conservative model of lexicalized PCFG
- Probabilistic conditioning is "top-down" like a regular PCFG (but actual computation is bottom-up)



Charniak (1997) example

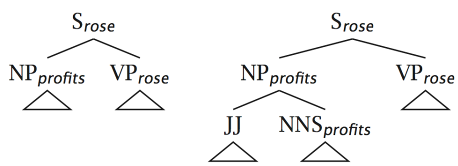


a. h = *profits*; c = NP

b. ph = *rose*; pc = S

c. $P(h|ph, c, pc)$

d. $P(r|h, c, pc)$



Lexicalization sharpens probabilities: rule expansion

- E.g., probability of different verbal complement frames (often called "subcategorizations")

Local Tree	*come*	*take*	*think*	*want*
VP → V	9.5%	2.6%	4.6%	5.7%
VP → V NP	1.1%	32.1%	0.2%	13.9%
VP → V PP	34.5%	3.1%	7.1%	0.3%
VP → V SBAR	6.6%	0.3%	73.0%	0.2%
VP → V S	2.2%	1.3%	4.8%	70.8%
VP → V NP S	0.1%	5.7%	0.0%	0.3%
VP → V PRT NP	0.3%	5.8%	0.0%	0.0%
VP → V PRT PP	6.1%	1.5%	0.2%	0.0%

Lexicalization sharpens probabilities: Predicting heads

"Bilexical probabilities"

- p(prices | n-plural) = .013
- p(prices | n-plural, NP) = .013
- p(prices | n-plural, NP, S) = .025
- p(prices | n-plural, NP, S, v-past) = .052
- p(prices | n-plural, NP, S, v-past, fell) = .146

Charniak (1997) linear interpolation/ shrinkage

$$\begin{aligned}\hat{P}(h|ph, c, pc) &= \lambda_1(e)P_{\text{MLE}}(h|ph, c, pc)\\ &\quad +\lambda_2(e)P_{\text{MLE}}(h|C(ph), c, pc)\\ &\quad +\lambda_3(e)P_{\text{MLE}}(h|c, pc) + \lambda_4(e)P_{\text{MLE}}(h|c)\end{aligned}$$

- $\lambda_i(e)$ is here a function of how much one would expect to see a certain occurrence, given the amount of training data, word counts, etc.
- $C(ph)$ is semantic class of parent headword
- Techniques like these for dealing with data sparseness are vital to successful model construction

Charniak (1997) shrinkage example

	$P(\text{prft}|\text{rose}, \text{NP}, \text{S})$	$P(\text{corp}|\text{prft}, \text{JJ}, \text{NP})$
$P(h|ph, c, pc)$	0	0.245
$P(h|C(ph), c, pc)$	0.00352	0.0150
$P(h|c, pc)$	0.000627	0.00533
$P(h|c)$	0.000557	0.00418

- Allows utilization of rich highly conditioned estimates, but smoothes when sufficient data is unavailable
- One can't just use MLEs: one commonly sees previously unseen events, which would have probability 0.

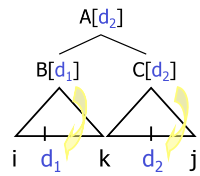
Sparseness & the Penn Treebank

- The Penn Treebank – 1 million words of parsed English WSJ – has been a key resource (because of the widespread reliance on supervised learning)
- But 1 million words is like nothing:
  - 965,000 constituents, but only 66 WHADJP, of which only 6 aren't *how much* or *how many*, but there is an infinite space of these
    - *How clever/original/incompetent (at risk assessment and evaluation) ...*
- Most of the probabilities that you would like to compute, you can't compute

Sparseness & the Penn Treebank (2)

- Many parse preferences depend on bilexical statistics: likelihoods of relationships between pairs of words (compound nouns, PP attachments, …)
- Extremely sparse, even on topics central to the WSJ:
  - *stocks plummeted* 2 occurrences
  - *stocks stabilized* 1 occurrence
  - *stocks skyrocketed* 0 occurrences
  - #*stocks discussed* 0 occurrences
- So far there has been very modest success in augmenting the Penn Treebank with extra unannotated materials or using semantic classes – once there is more than a little annotated training data.
  - Cf. Charniak 1997, Charniak 2000; but see McClosky et al. 2006 [this recent self-training work is quite successful!]

Complexity of lexicalized PCFG parsing

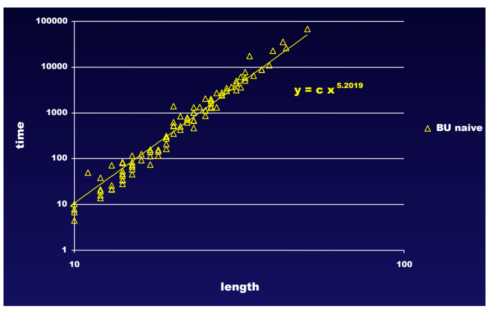


Time charged :

- i, k, j ⇒ n^3
- A, B, C ⇒ g^3
- Naively, g becomes huge
- d_1, d_2 ⇒ n^2

Running time is $O(g^3 \times n^5)$!!

Complexity of exhaustive lexicalized PCFG parsing



Complexity of lexicalized PCFG parsing

- Work such as Collins (1997) and Charniak (1997) *is* $O(n^5)$ – but uses heuristic search to be fast in practice
- Eisner and Satta (2000, etc.) have explored various ways to parse more restricted classes of bilexical grammars in $O(n^4)$ or $O(n^3)$ time
  - Neat algorithmic stuff!!!
  - See example later from dependency parsing

Refining the node expansion probabilities

- Charniak (1997) expands each phrase structure tree in a single step.
- This is good for capturing dependencies between child nodes
- But it is bad because of data sparseness.
- A pure dependency, one child at a time, model is worse.
- But one can do better by in between models, such as generating the children as a Markov process on both sides of the head (Collins 1997; Charniak 2000)
  - Cf. the accurate unlexicalized parsing discussion

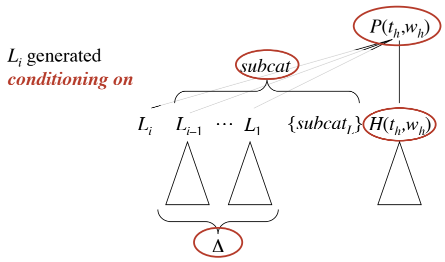
Collins (1997, 1999); Bikel (2004)

- Collins (1999): also a generative model
- Underlying lexicalized PCFG has rules of form

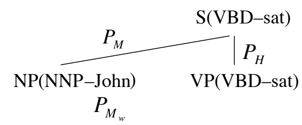
$$P \rightarrow L_j L_{j-1} \cdots L_1 H R_1 \cdots R_{k-1} R_k$$

- A more elaborate set of grammar transforms and factorizations to deal with data sparseness and interesting linguistic properties
- Each child is generated in turn: given *P* has been generated, generate *H*, then generate modifying nonterminals from head-adjacent outward with some limited conditioning

Overview of Collins' Model



Modifying nonterminals generated in two steps



Smoothing for head words of modifying nonterminals

Back-off level	$P_{M_w}(w_M \mid \ldots)$
0	M_i, t_{M_i}, coord, punc, $P, H, w_h, t_h, \Delta_M, subcat_{side}$
1	M_i, t_{M_i}, coord, punc, $P, H, t_h, \Delta_M, subcat_{side}$
2	t_{M_i}

- Other parameter classes have similar or more elaborate backoff schemes

Collins model … and linguistics

- Collins had 3 generative models: Models 1 to 3
- Especially as you work up from Model 1 to 3, significant linguistic modeling is present:
  - Distance measure: favors close attachments
    - Model is sensitive to punctuation
  - Distinguish base NP from full NP with post-modifiers
  - Coordination feature
  - Mark gapped subjects
  - Model of subcategorization; arguments vs. adjuncts
  - Slash feature/gap threading treatment of displaced constituents
    - Didn't really get clear gains from this last one.

Bilexical statistics: Is use of maximal context of P_{M_w} useful?

- Collins (1999): "Most importantly, the model has parameters corresponding to dependencies between pairs of headwords."
- Gildea (2001) reproduced Collins' Model 1 (like regular model, but no subcats)
  - Removing maximal back-off level from P_{M_w} resulted in only 0.5% reduction in F-measure
  - Gildea's experiment somewhat unconvincing to the extent that his model's performance was lower than Collins' reported results

Choice of heads

- If not bilexical statistics, then surely choice of heads is important to parser performance…
- Chiang and Bikel (2002): parsers performed decently even when all head rules were of form "if parent is X, choose left/rightmost child"
- Parsing engine in Collins Model 2–emulation mode: LR 88.55% and LP 88.80% on §00 (sent. len. ≤40 words)
  - compared to LR 89.9%, LP 90.1%

Use of maximal context of P_{M_w}
[Bikel 2004]

	LR	LP	CBs	0 CBs	≤2 CBs
Full model	89.9	90.1	0.78	68.8	89.2
No bigrams	89.5	90.0	0.80	68.0	88.8

Performance on §00 of Penn Treebank
on sentences of length ≤40 words

Use of maximal context of P_{M_w}

Back-off level	Number of accesses	Percentage
0	3,257,309	1.49
1	24,294,084	11.0
2	191,527,387	87.4
Total	219,078,780	100.0

Number of times parsing engine was able to deliver a probability for the various back-off levels of the mod-word generation model, P_{M_w}, when testing on §00 having trained on §§02–21

Bilexical statistics *are* used often
[Bikel 2004]

- The 1.49% use of bilexical dependencies suggests they don't play much of a role in parsing
- But the parser pursues many (very) incorrect theories
- So, instead of asking how often the decoder can use bigram probability *on average*, ask how often *while pursuing its top-scoring theory*
- Answering question by having parser *constrain-parse* its own output
  - train as normal on §§02–21
  - parse §00
  - feed parse trees as *constraints*
- Percentage of time parser made use of bigram statistics shot up to **28.8%**
- So, used often, but use barely affect overall parsing accuracy
- Exploratory Data Analysis suggests explanation
  - distributions that include head words are usually sufficiently similar to those that do not as to make almost no difference in terms of accuracy

Charniak (2000) NAACL: A Maximum-Entropy-Inspired Parser

- There was nothing maximum entropy about it. It was a cleverly smoothed generative model
- Smoothes estimates by smoothing ratio of conditional terms (which are a bit like maxent features):

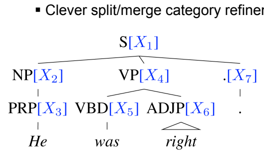
$$\frac{P(t\,|\,l, l_p, t_p, l_g)}{P(t\,|\,l, l_p, t_p)}$$

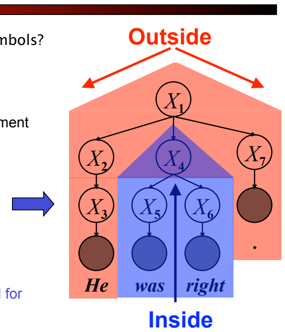
- Biggest improvement is actually that generative model predicts head tag first and then does P(*w*|*t*,…)
  - Like Collins (1999)
- Markovizes rules similarly to Collins (1999)
- Gets 90.1% LP/LR F score on sentences ≤ 40 wds

Petrov and Klein (2006): Learning Latent Annotations

Can you automatically find good symbols?

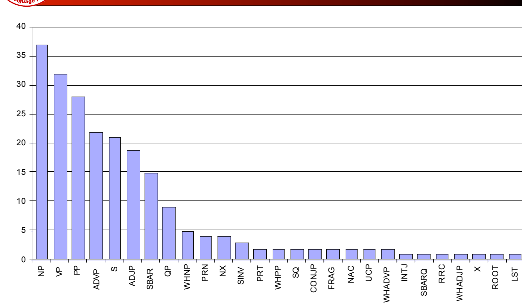
- Brackets are known
- Base categories are known
- Induce subcategories
- Clever split/merge category refinement





EM algorithm, like Forward-Backward for HMMs, but constrained by tree.

Number of phrasal subcategories



POS tag splits, commonest words: effectively a class-based model

- Proper Nouns (NNP):

NNP-14	Oct.	Nov.	Sept.
NNP-12	John	Robert	James
NNP-2	J.	E.	L.
NNP-1	Bush	Noriega	Peters
NNP-15	New	San	Wall
NNP-3	York	Francisco	Street

- Personal pronouns (PRP):

PRP-0	It	He	I
PRP-1	it	he	they
PRP-2	it	them	him

The Latest Parsing Results…

Parser	*F1* *≤ 40 words*	*F1* *all words*
Klein & Manning unlexicalized 2003	86.3	85.7
Matsuzaki et al. simple EM latent states 2005	86.7	86.1
Charniak generative ("maxent inspired") 2000	90.1	89.5
Petrov and Klein NAACL 2007	90.6	90.1
Charniak & Johnson discriminative reranker 2005	92.0	91.4

Statistical parsing inference: The General Problem

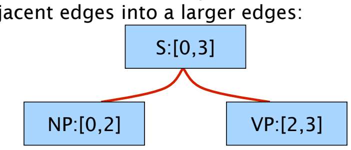
- Someone gives you a PCFG *G*
- For any given sentence, you might want to:
  - Find the best parse according to *G*
  - Find a bunch of reasonably good parses
  - Find the total probability of all parses licensed by *G*
- Techniques:
  - CKY, for best parse; can extend it:
    - To *k*-best: naively done, at high space and time cost – k^2 time/*k* space cost, but there are cleverer algorithms! (Huang and Chiang 2005: http://www.cis.upenn.edu/~lhuang3/huang-iwpt.pdf)
    - To all parses, summed probability: the inside algorithm
  - Beam search
  - Agenda/chart-based search

} Mainly useful if just want the best parse

Parse as search definitions

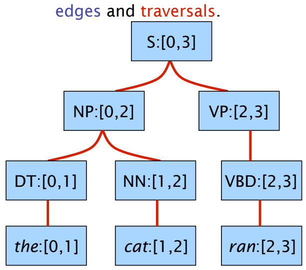
- Grammar symbols: S, NP, @S->NP_
- Parse items/edges represent a grammar symbol over a span:

- Backtraces/traversals represent the combination of adjacent edges into a larger edges:

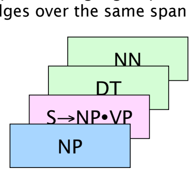


Parse trees and parse triangles

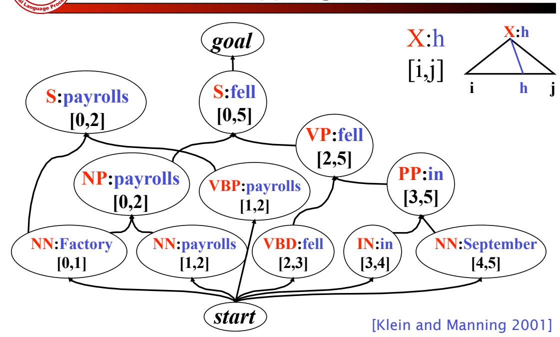
- A parse tree can be viewed as a collection of edges and traversals.



- A parse triangle groups edges over the same span

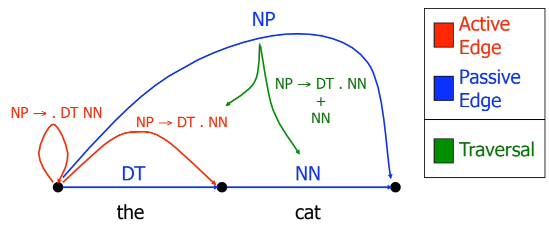


Parsing as search: The parsing directed B-hypergraph



[Klein and Manning 2001]

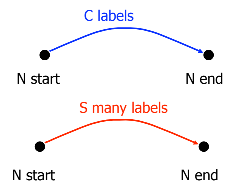
Chart example: classic picture



Earley dotted rules

Space and Time Bounds

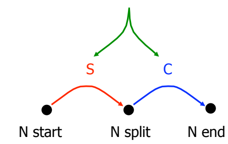
Space = O(Edges)



$\leq CN^2 + SN^2$

$= O(SN^2)$

Time = O(Traversals)

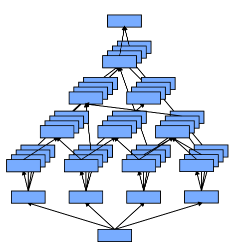


$\leq SCN^3$

$= O(SCN^3)$

CKY Parsing

- In CKY parsing, we visit edges tier by tier:



- Guarantees correctness by working inside-out.
- Build all small bits before any larger bits that could possibly require them.
- Exhaustive: the goal is in the last tier!

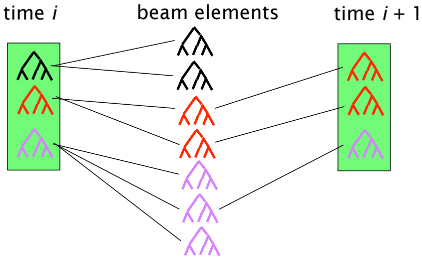
Beam Search

- 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

Beam Search

- Time-synchronous beam search

Beam at time *i* | Successors of beam elements | Beam at time *i* + 1



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 (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")

Agenda-based parsing

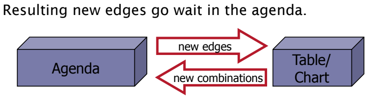
- For general grammars
- Start with a table for recording δ(X,i,j)
  - Records the best score of a parse of X over [i,j]
    - If the scores are negative log probabilities, then entries start at ∞ and small is good
    - This can be a sparse or a dense map
    - Again, you may want to record backtraces (traversals) as well, like CKY
- Step 1: Initialize with the sentence and lexicon:
  - For each word w and each tag t
    - Set δ(X,i,i) = lex.score(w,t)

Agenda-based parsing

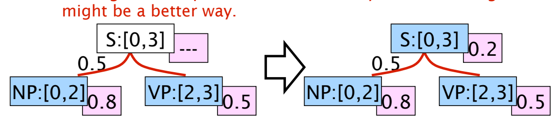
- Keep a list of edges called an agenda
  - Edges are triples [X,i,j]
  - The agenda is a priority queue
- Every time the score of some δ(X,i,j) improves (i.e. gets lower):
  - Stick the edge [X,i,j]-score into the agenda
  - (Update the backtrace for δ(X,i,j) if your storing them)

Agenda-Based Parsing

- The agenda is a holding zone for edges.
- Visit edges by some ordering policy.
  - Combine edge with already-visited edges.
  - Resulting new edges go wait in the agenda.



- We might revisit parse items: A new way to form an edge might be a better way.



Agenda-based parsing

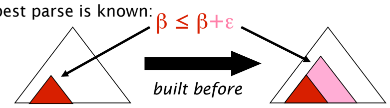
- Step II: While agenda not empty
  - Get the "next" edge [X,i,j] from the agenda
  - Fetch all compatible neighbors [Y,j,k] or [Z,k,i]
    - Compatible means that there are rules A→X Y or B→ Z X
  - Build all parent edges [A,i,k] or [B,k,j] found
    - δ(A,i,k) ≤ δ(X,i,j) + δ(Y,j,k) + P(A→X Y)
    - If we've improved δ(A,i,k), then stick it on the agenda
  - Also project unary rules:
    - Fetch all unary rules A→X, score [A,i,j] built from this rule on [X,i,j] and put on agenda if you've improved δ(A,i,k)
- When do we know we have a parse for the root?

Agenda-based parsing

- Open questions:
  - Agenda priority: What did "next" mean?
  - Efficiency: how do we do as little work as possible?
  - Optimality: how do we know when we find the best parse of a sentence?
- If we use δ(X,i,j) as the priority:
  - Each edge goes on the agenda at most once
  - When an edge pops off the agenda, its best parse is known (why?)
  - This is basically uniform cost search (i.e., Dijkstra's algorithm). [Cormen, Leiserson, and Rivest 1990; Knuth 1970]

Uniform-Cost Parsing

- We want to work on good parses inside-out.
  - CKY does this synchronously, by tiers.
  - Uniform-cost does it asynchronously, ordering edges by their best known parse score.
- Why best parse is known:



  - Adding structure incurs probability cost.
  - Trees have lower probability than their sub-parts.
  - The best-scored edge in the agenda cannot be waiting on any of its sub-edges.
  - We never have to propagate. We don't explore truly useless edges.

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?

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

Speed: build promising edges 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 edges on the agenda until you're sure you've seen their best parse.