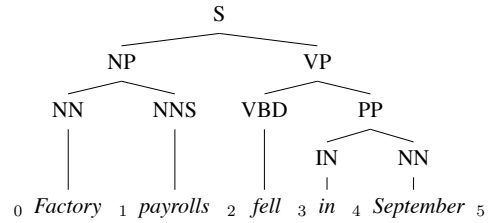
Chapter 5

Constituent-Context Models

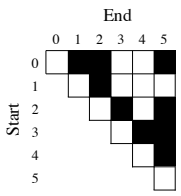
5.1 Previous Work

In contrast with the relative success of word-class learning methods, induction of syntax has proven to be extremely challenging. Some of the earliest and most important signs of discouragement from statistical parameter search methods were the results of Lari and Young (1990). Their work showed that even simple artificial grammars could not be reliably recovered using EM over the space of PCFGs (using the inside-outside algorithm: see Manning and Schütze (1999) for an introduction). The problem wasn't with the model family: Charniak (1996) showed that a maximum-likelihood PCFG read off of a treebank could parse reasonably well, and most high-performance parsers have, strictly speaking, been in the class of PCFG parsing. Therefore the problem was either with the use of EM as a search procedure or with some mismatch between data likelihood and grammar quality. Either way, their work showed that simple grammars were hard to recover in a fully unsupervised manner.

Carroll and Charniak (1992) tried the PCFG induction approach on natural language data, again with discouraging results. They used a structurally restricted PCFG in which the terminal symbols were parts-of-speech and the non-terminals were part-of-speech projections. That is, for every part-of-speech X there was a non-terminal $\overline{\text{X}}$, with the rewrites restricted to the forms $\overline{\text{X}} \rightarrow \overline{\text{X}}\ \overline{\text{Y}}$ and $\overline{\text{X}} \rightarrow \overline{\text{Y}}\ \overline{\text{X}}$ (plus unary terminal rewrites of the form $\overline{\text{X}} \rightarrow$ X). These configurations can be thought of as head-argument attachments, where X is



(a)



(b)

Span	Label	Constituent	Context
⟨0,5⟩	S	NN NNS VBD IN NN	◇ – ◇
⟨0,2⟩	NP	NN NNS	◇ – VBD
⟨2,5⟩	VP	VBD IN NN	NNS – ◇
⟨3,5⟩	PP	IN NN	VBD – ◇
⟨0,1⟩	NN	NN	◇ – NNS
⟨1,2⟩	NNS	NNS	NN – VBD
⟨2,3⟩	VBD	VBD	NNS – IN
⟨3,4⟩	IN	IN	VBD – NN
⟨4,5⟩	NN	NNS	IN – ◇

(c)

Figure 5.1: Parses, bracketings, and the constituent-context representation for the sentence, "Factory payrolls fell in September." Shown are (a) an example parse tree, (b) its associated bracketing, and (c) the yields and contexts for each constituent span in that bracketing. Distituent yields and contexts are not shown, but *are* modeled.

the head. In fact, trees in this grammar are isomorphic to dependency trees which specify the attachment order for heads with multiple arguments (Miller 1999). The hope was that, while the symbols in an arbitrary PCFG do not have any *a priori* meaning or structural role, symbols in this dependency grammar are not structurally symmetric – each one is anchored to a specific terminal symbol. Carroll and Charniak describe experiments where many such grammars were weighted randomly, then re-estimated using EM. The resulting grammars exhibited wide variance in the structures learned and in the data likelihood found. Parsing performance was consistently poor (according to their qualitative evaluation). Their conclusion was that the blame lay with the structure search problem: EM is a local maximization procedure, and each initial PCFG converged to a different final grammar. Regardless of the cause, the results did not suggest that PCFG induction was going to be straightforwardly effective.

Other related parameter search work is discussed in section 6.1.2, but it is worth further considering the Carroll and Charniak experiments and results here. One important advantage of their formulation (that they did not exploit) is that random initial rule weights are not actually needed. In the case of unrestricted binary-branching PCFGs, such as with the Lari and Young (1990) experiments, one considers a full binary grammar over symbols $\{\mathrm{X}_i\}$. If all rules $\mathrm{X}_i \rightarrow \mathrm{X}_j\ \mathrm{X}_k$ have exactly equal initial probability, that initial parameter vector will be an unstable symmetric fixed point for EM. Therefore random noise is required for symmetry-breaking, to get the optimization started. That is not the case for the Carroll and

Charniak grammars. While the parameter space is certainly riddled with local maxima, and therefore the initial grammar weights do matter, there is an obvious uniform starting point, where all rules have equal starting probability. Beginning from that uniform initializer, EM will find some solution which we might hope will correspond to a higher quality grammar than most random initializations produce. This hope is borne out in practice: as figure 5.4 shows under the name DEP-PCFG, their method substantially outperforms a random baseline. It does not break the right-branching baseline, however, and we can ask why that might be. One cause is certainly that the grammar itself is representationally lacking; we will discuss this further in chapter 6. Section 5.3.6 discusses another possible issue: a flat grammar initialization gives rise to a very un-language-like posterior over trees.

The distributional clustering of words (chapter 3) has proven remarkably robust for discovering patterns which at least broadly approximate classical parts-of-speech. It is therefore very appealing to try to extend linear distributional techniques to levels of syntax higher than word-classes. Recall the left column of figure 3.3, which shows the most similar tag sequences according to the Jensen-Shannon divergence of their local linear tag signatures. This list makes one optimistic that constituent sequences with very similar contexts will tend to be of the same constituent type. For example, the top three pairs are noun groups, prepositional phrases, and determiner-carrying noun phrases. The subsequent examples include more correct noun and prepositional phrase pairs, with some possessive constructions and verb phrases scattered among them. Indeed, the task of taking constituent sequences and clustering them into groups like noun-phrases and verb phrases is not much harder than clustering words into word classes. The problem is that to produce lists like these, we need to know which subspans of each sentence are constituents. If we simply consider all subspans of the sentences in our corpus, most sequence tokens will not be constituent tokens. The right column of figure 3.2 shows the sequence pairs with most similar contexts, using all subspans instead of constituent subspans. Again we see pairs of similar constituents, like the first pair of proper noun phrases. However, we also see examples like the second pair, which are two non-constituent sequences. It's no surprise these non-constituent, or *distituent*, pairs have similar context distributions – if we had to classify them, they are in some sense similar. But a successful grammar induction system must somehow learn which sequence types should be regularly used in building trees, and which

should not. That is, we need to form coherent tree-structured analyses, and distributional clustering of sequences, robust though it may be, will not give us trees.

One way to get around this limitation of distributional clustering is to first group sequences into types by signature similarity, then differentiate the "good" and "bad" constituent types by some other mechanism. A relatively successful approach along these lines is described in Clark (2001a). Clark first groups sequence types, then uses a mutual information criterion to filter constituents from distituents. The good sequences are then used to build up a PCFG according to a MDL measure. The experimental results of Clark's system are discussed later in this chapter, but the overall parsing performance is rather low because the discovered grammars are extremely sparse.

5.2 A Generative Constituent-Context Model

In this chapter, we describe a model which is designed to combine the robustness of distributional clustering with the coherence guarantees of parameter search. It is specifically intended to produce a more felicitous search space by removing as much hidden structure as possible from the syntactic analyses. The fundamental assumption is a much weakened version of a classic linguistic constituency tests (Radford 1988): constituents appear in constituent context. A particular linguistic phenomenon that the system exploits is that long constituents often have short, common equivalents, or *proforms*, which appear in similar contexts and whose constituency is easily discovered (or guaranteed). Our model is designed to transfer the constituency of a sequence directly to its containing context, which is intended to then pressure new sequences that occur in that context into being parsed as constituents in the next round. The model is also designed to exploit the successes of distributional clustering, and can equally well be viewed as doing distributional clustering in the presence of no-overlap constraints.

5.2.1 Constituents and Contexts

Unlike a PCFG, our model describes all contiguous subsequences of a sentence (*spans*), including empty spans, whether they are constituents or distituents. A span encloses a sequence of terminals, or *yield*, α, such as DT JJ NN. A span occurs in a *context* x, such as $\diamond$–VBZ, where x is the ordered pair of preceding and following terminals ($\diamond$ denotes a sentence boundary). A *bracketing* of a sentence is a boolean matrix B, which indicates which spans are constituents and which are not. Figure 5.1 shows a parse of a short sentence, the bracketing corresponding to that parse, and the labels, yields, and contexts of its constituent spans.

Figure 5.2 shows several bracketings of the sentence in figure 5.1. A bracketing B of a sentence is *non-crossing* if, whenever two spans cross, at most one is a constituent in B. A non-crossing bracketing is *tree-equivalent* if the size-one terminal spans and the full-sentence span are constituents, and all size-zero spans are distituents. Figure 5.2(a) and (b) are tree-equivalent. Tree-equivalent bracketings B correspond to (unlabeled) trees in the obvious way. A bracketing is *binary* if it corresponds to a binary tree. Figure 5.2(b) is binary. We will induce trees by inducing tree-equivalent bracketings.

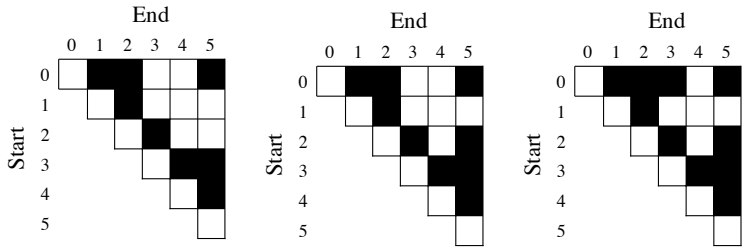
Our generative model over sentences S has two phases. First, we choose a bracketing B according to some distribution $\mathrm{P}(B)$ and then generate the sentence given that bracketing:

$$\mathrm{P}(S, B) = \mathrm{P}(B)\mathrm{P}(S|B)$$

Given B, we fill in each span independently. The context and yield of each span are independent of each other, and generated conditionally on the constituency B_{ij} of that span.

$$\begin{aligned} \mathrm{P}(S|B) &= \prod_{\langle i,j\rangle \in spans(S)} \mathrm{P}(\alpha_{ij}, x_{ij}|B_{ij}) \\ &= \prod_{\langle i,j\rangle} \mathrm{P}(\alpha_{ij}|B_{ij})\mathrm{P}(x_{ij}|B_{ij}) \end{aligned}$$

The distribution $\mathrm{P}(\alpha_{ij}|B_{ij})$ is a pair of multinomial distributions over the set of all possible yields: one for constituents ($B_{ij} = c$) and one for distituents ($B_{ij} = d$). Similarly for $\mathrm{P}(x_{ij}|B_{ij})$ and contexts. The marginal probability assigned to the sentence S is given by summing over all possible bracketings of S: $\mathrm{P}(S) = \sum_B \mathrm{P}(B)\mathrm{P}(S|B)$. Note that this is



(a) Tree-equivalent (b) Binary (c) Crossing

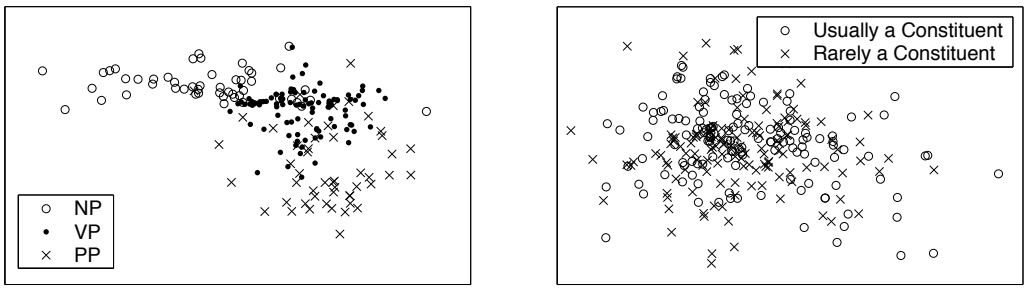
Figure 5.2: Three bracketings of the sentence "Factory payrolls fell in September." Constituent spans are shown in black. The bracketing in (b) corresponds to the binary parse in figure 5.1; (a) does not contain the ⟨2,5⟩ VP bracket, while (c) contains a ⟨0,3⟩ bracket crossing that VP bracket.

a more severe set of independence assumptions than, say, in a naive-bayes model. There, documents positions are filled independently, and the result can easily be an ungrammatical document. Here, the result need not even be a structurally consistent sentence.[1]

To induce structure, we run EM over this model, treating the sentences S as observed and the bracketings B as unobserved. The parameters Θ of the model are the constituency-conditional yield and context distributions $\mathrm{P}(\alpha|b)$ and $\mathrm{P}(x|b)$. If $\mathrm{P}(B)$ is uniform over all (possibly crossing) bracketings, then this procedure will be equivalent to soft-clustering with two equal-prior classes.

There is reason to believe that such soft clusterings alone will not produce valuable distinctions, even with a significantly larger number of classes. The distituents must necessarily outnumber the constituents, and so such distributional clustering will result in mostly distituent classes. Clark (2001a) finds exactly this effect, and must resort to a filtering heuristic to separate constituent and distituent clusters. To underscore the difference between the bracketing and labeling tasks, consider figure 5.3. In both plots, each point is a frequent tag sequence, assigned to the (normalized) vector of its context frequencies. Each plot has been projected onto the first two principal components of its respective data set. The left

[1] Viewed as a model generating *sentences*, this model is deficient, placing mass on yield and context choices which will not tile into a valid sentence, either because specifications for positions conflict or because yields of incorrect lengths are chosen. We might in principle renormalize by dividing by the mass placed on proper sentences and zeroing the probability of improper bracketings. In practice, there does not seem to be an easy way to carry out this computation.



(a) Constituent Types (b) Constituents vs. Distituents

Figure 5.3: Clustering vs. detecting constituents. The most frequent yields of (a) three constituent types and (b) constituents and distituents, as context vectors, projected onto their first two principal components. Clustering is effective at labeling, but not detecting, constituents.

plot shows the most frequent sequences of three constituent types. Even in just two dimensions, the clusters seem coherent, and it is easy to believe that they would be found by a clustering algorithm in the full space. On the right, sequences have been labeled according to whether their occurrences are constituents more or less of the time than a cutoff (of 0.2). The distinction between constituent and distituent seems much less easily discernible.

We can turn what at first seems to be distributional clustering into tree induction by confining $\mathrm{P}(B)$ to put mass only on tree-equivalent bracketings. In particular, consider $\mathrm{P}_{\mathrm{bin}}(B)$ which is uniform over binary bracketings and zero elsewhere. If we take this bracketing distribution, then when we sum over data completions, we will only involve bracketings which correspond to valid binary trees. This restriction is the basis for the next algorithm.

5.2.2 The Induction Algorithm

We now essentially have our induction algorithm. We take $\mathrm{P}(B)$ to be $\mathrm{P}_{\mathrm{bin}}(B)$, so that all binary trees are equally likely. We then apply the EM algorithm:

E-Step: Find the conditional completion likelihoods $\mathrm{P}(B|S,\Theta)$ according to the current Θ.

M-Step: Fix $\mathrm{P}(B|S,\Theta)$ and find the Θ' which maximizes $\sum_B \mathrm{P}(B|S,\Theta)\log \mathrm{P}(S,B|\Theta')$.

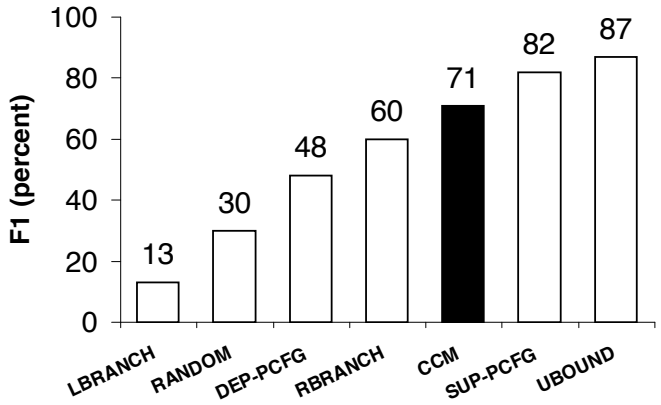


Figure 5.4: Bracketing F_1 for various models on the WSJ10 data set.

The completions (bracketings) cannot be efficiently enumerated, and so a cubic dynamic program similar to the inside-outside algorithm is used to calculate the expected counts of each yield and context, both as constituents and distituents (see the details in appendix A.1). Relative frequency estimates (which are the ML estimates for this model) are used to set Θ'.

5.3 Experiments

The experiments that follow used the WSJ10 data set, as described in chapter 2, using the alternate unlabeled metrics described in section 2.2.5, with the exception of figure 5.15 which uses the standard metrics, and figure 5.6 which reports numbers given by the EVALB program. The basic experiments do not label constituents. An advantage to having only a single constituent class is that it encourages constituents of one type to be proposed even when they occur in a context which canonically holds another type. For example, NPs and PPs both occur between a verb and the end of the sentence, and they can transfer constituency to each other through that context.

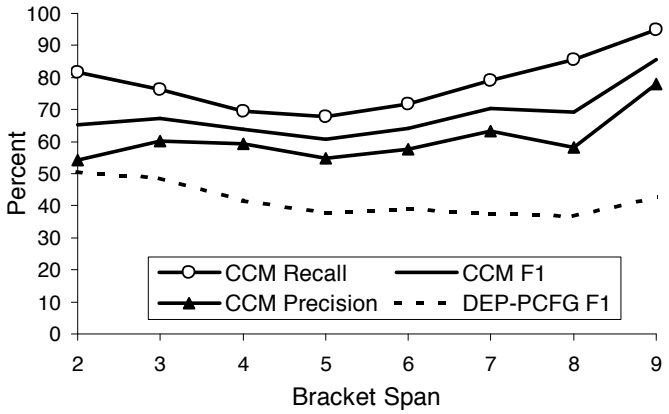


Figure 5.5: Scores for CCM-induced structures by span size. The drop in precision for span length 2 is largely due to analysis inside NPs which is omitted by the treebank. Also shown is F_1 for the induced PCFG. The PCFG shows higher accuracy on small spans, while the CCM is more even.

Figure 5.4 shows the F_1 score for various methods of parsing. RANDOM chooses a tree uniformly at random from the set of binary trees.[2] This is the unsupervised baseline. DEP-PCFG is the result of duplicating the experiments of Carroll and Charniak (1992), using EM to train a dependency-structured PCFG. LBRANCH and RBRANCH choose the left- and right-branching structures, respectively. RBRANCH is a frequently used baseline for *supervised* parsing, but it should be stressed that it encodes a significant fact about English structure, and an induction system need not beat it to claim a degree of success. CCM is our system, as described above. SUP-PCFG is a supervised PCFG parser trained on a 90-10 split of this data, using the treebank grammar, with the Viterbi parse right-binarized.[3] UBOUND is the upper bound of how well a binary system can do against the treebank sentences, which are generally flatter than binary, limiting the maximum precision.

CCM is doing quite well at 71.1%, substantially better than right-branching structure. One common issue with grammar induction systems is a tendency to chunk in a bottom-up fashion. Especially since the CCM does not model recursive structure explicitly, one might be concerned that the high overall accuracy is due to a high accuracy on short-span

[2]This is different from making random parsing decisions, which gave a higher score of 35%.

[3]Without post-binarization, the F_1 score was 88.9.

System	UP	UR	F_1	CB
EMILE	51.6	16.8	25.4	**0.84**
ABL	43.6	35.6	39.2	2.12
CDC-40	53.4	34.6	42.0	1.46
RBRANCH	39.9	46.4	42.9	2.18
CCM	**55.4**	**47.6**	**51.2**	1.45

Figure 5.6: Comparative ATIS parsing results.

constituents. Figure 5.5 shows that this is not true. Recall drops slightly for mid-size constituents, but longer constituents are as reliably proposed as short ones. Another effect illustrated in this graph is that, for span 2, constituents have low precision for their recall. This contrast is primarily due to the single largest difference between the system's induced structures and those in the treebank: the treebank does not parse into NPs such as DT JJ NN, while our system does, and generally does so correctly, identifying $\overline{\text{N}}$ units like JJ NN. This overproposal drops span-2 precision. In contrast, figure 5.5 also shows the F_1 for DEP-PCFG, which does exhibit a drop in F_1 over larger spans.

The top row of figure 5.8 shows the recall of non-trivial brackets, split according the brackets' labels in the treebank. Unsurprisingly, NP recall is highest, but other categories are also high. Because we ignore trivial constituents, the comparatively low S represents *only* embedded sentences, which are somewhat harder even for supervised systems.

To facilitate comparison to other recent work, figure 5.6 shows the accuracy of our system when trained on the same WSJ data, but tested on the ATIS corpus, and evaluated according to the EVALB program. EMILE and ABL are lexical systems described in (van Zaanen 2000, Adriaans and Haas 1999), both of which operate on minimal pairs of sentences, deducing constituents from regions of variation. CDC-40, from (Clark 2001a), reflects training on much more data (12M words), and is describe in section 5.1. The F_1 numbers are lower for this corpus and evaluation method.[4] Still, CCM beats not only RBRANCH (by 8.3%), but the next closest unsupervised system by slightly more.

[4]The primary cause of the lower F_1 is that the ATIS corpus is replete with span-one NPs; adding an extra bracket around *all* single words raises our EVALB recall to 71.9; removing all unaries from the ATIS gold standard gives an F_1 of 63.3%.

Rank	Overproposed	Underproposed
1	JJ NN	NNP POS
2	MD VB	TO CD CD
3	DT NN	NN NNS
4	NNP NNP	NN NN
5	RB VB	TO VB
6	JJ NNS	IN CD
7	NNP NN	NNP NNP POS
8	RB VBN	DT NN POS
9	IN NN	RB CD
10	POS NN	IN DT

Figure 5.7: Constituents most frequently over- and under-proposed by our system.

5.3.1 Error Analysis

Parsing figures can only be a component of evaluating an unsupervised induction system. Low scores may indicate systematic alternate analyses rather than true confusion, and the Penn treebank is a sometimes arbitrary or even inconsistent gold standard. To give a better sense of the kinds of errors the system is or is not making, we can look at which sequences are most often overproposed, or most often underproposed, compared to the treebank parses.

Figure 5.7 shows the 10 most frequently over- and under-proposed sequences. The system's main error trends can be seen directly from these two lists. It forms MD VB verb groups systematically, and it attaches the possessive particle to the right, like a determiner, rather than to the left.[5] It provides binary-branching analyses within NPs, normally resulting in correct extra $\overline{\text{N}}$ constituents, like JJ NN, which are not bracketed in the treebank. More seriously, it tends to attach post-verbal prepositions to the verb and gets confused by long sequences of nouns. A significant improvement over some earlier systems (both ours and other researchers') is the absence of subject-verb groups, which disappeared when we switched to $\mathrm{P}_{\mathrm{split}}(B)$ for initial completions (see section 5.3.6); the more balanced subject-verb analysis had a substantial combinatorial advantage with $\mathrm{P}_{\mathrm{bin}}(B)$.

[5]Linguists have at times argued for both analyses: Halliday (1994) and Abney (1987), respectively.

5.3.2 Multiple Constituent Classes

We also ran the system with multiple constituent classes, using a slightly more complex generative model in which the bracketing generates a labeling L (a mapping from spans to label classes C) which then generates the constituents and contexts.

$$\mathrm{P}(S, L, B) = \mathrm{P}(B)\mathrm{P}(L|B)\mathrm{P}(S|L)$$

$$\mathrm{P}(L|B) = \prod_{\langle i,j\rangle \in spans(S)} \mathrm{P}(L_{ij}|B_{ij})$$

$$\begin{aligned}\mathrm{P}(S|L) &= \prod_{\langle i,j\rangle \in spans(S)} \mathrm{P}(\alpha_{ij}, x_{ij}|L_{ij}) \\ &= \prod_{\langle i,j\rangle} \mathrm{P}(\alpha_{ij}|L_{ij})\mathrm{P}(x_{ij}|L_{ij})\end{aligned}$$

The set of labels for constituent spans and distituent spans are forced to be disjoint, so $\mathrm{P}(L_{ij}|B_{ij})$ is given by

$$\mathrm{P}(L_{ij}|B_{ij}) = \begin{cases} 1 & \text{if } B_{ij} = \textit{false} \wedge L_{ij} = d \\ 0 & \text{if } B_{ij} = \textit{false} \wedge L_{ij} \neq d \\ 0 & \text{if } B_{ij} = \textit{true} \wedge L_{ij} = d \\ 1/|C-1| & \text{if } B_{ij} = \textit{true} \wedge L_{ij} \neq d \end{cases}$$

where d is a distinguished distituent-only label, and the other labels are sampled uniformly at each constituent span.

Intuitively, it seems that more classes should help, by allowing the system to distinguish different types of constituents and constituent contexts. However, it seemed to slightly hurt parsing accuracy overall. Figure 5.8 compares the performance for 2 versus 12 classes; in both cases, only one of the classes was allocated for distituents. Overall F_1 dropped very slightly with 12 classes, but the category recall numbers indicate that the errors shifted around substantially. PP accuracy is lower, which is not surprising considering that PPs

Classes	Tags	Precision	Recall	F_1	NP Recall	PP Recall	VP Recall	S Recall
2	Treebank	**63.8**	**80.2**	**71.1**	**83.4**	**78.5**	78.6	40.7
12	Treebank	63.6	80.0	70.9	82.2	59.1	82.8	57.0
2	Induced	56.8	71.1	63.2	52.8	56.2	**90.0**	**60.5**

Figure 5.8: Scores for the 2- and 12-class model with Treebank tags, and the 2-class model with induced tags.

Class 0		Class 1	Class 2	Class 3	Class 4	Class 5	Class 6
NNP NNP	NN VBD	DT NN	NNP NNP	CD CD	VBN IN	MD VB	JJ NN
NN IN	NN NN	JJ NNS	NNP NNP NNP	CD NN	JJ IN	MD RB VB	JJ NNS
IN DT	NNS VBP	DT NNS	CC NNP	IN CD CD	DT NN	VBN IN	JJ JJ NN
DT JJ	NNS VBD	DT JJ NN	POS NN	CD NNS	JJ CC	WDT VBZ	CD NNS
NN VBZ	TO VB	NN NNS	NNP NNP NNP NNP	CD CD IN CD CD	DT JJ NN	JJ IN	NNP NN

Figure 5.9: Most frequent members of several classes found.

tend to appear rather optionally and in contexts in which other, easier categories also frequently appear. On the other hand, embedded sentence recall is substantially higher, possibly because of more effective use of the top-level sentences which occur in the context ◇–◇.

The classes found, as might be expected, range from clearly identifiable to nonsense. Note that simply directly clustering all sequence types into 12 categories based on their local linear distributional signatures produced almost entirely the latter, with clusters representing various distituent types. Figure 5.9 shows several of the 12 classes. Class 0 is the model's distituent class. Its most frequent members are a mix of obvious distituents (IN DT, DT JJ, IN DT, NN VBZ) and seemingly good sequences like NNP NNP. However, there are many sequences of 3 or more NNP tags in a row, and not all adjacent pairs can possibly be constituents at the same time. Class 1 is mainly common NP sequences, class 2 is proper NPs, class 3 is NPs which involve numbers, and class 6 is $\overline{\text{N}}$ sequences, which tend to be linguistically right but unmarked in the treebank. Class 4 is a mix of seemingly good NPs, often from positions like VBZ–NN where they were *not* constituents, and other sequences that share such contexts with otherwise good NP sequences. This is a danger of not jointly modeling yield and context, and of not modeling any kind of recursive structure: our model cannot learn that a sequence is a constituent only in certain contexts (the best we can hope for is that such contexts will be learned as strong distituent contexts). Class 5 is mainly composed of verb phrases and verb groups. No class corresponded neatly to PPs: perhaps because they have no signature contexts. The 2-class model is effective at identifying them

only because they share contexts with a range of other constituent types (such as NPs and VPs).

5.3.3 Induced Parts-of-Speech

A reasonable criticism of the experiments presented so far, and some other earlier work, is that we assume treebank part-of-speech tags as input. This criticism could be two-fold. First, state-of-the-art supervised PCFGs do not perform nearly so well with their input delexicalized. We may be reducing data sparsity and making it easier to see a broad picture of the grammar, but we are also limiting how well we can possibly do. It is certainly worth exploring methods which supplement or replace tagged input with lexical input. However, we address here the more serious criticism: that our results stem from clues latent in the treebank tagging information which are conceptually posterior to knowledge of structure. For instance, some treebank tag distinctions, such as particle (RP) vs. preposition (IN) or predeterminer (PDT) vs. determiner (DT) or adjective (JJ), could be said to import into the tag set distinctions that can only be made syntactically.

To show results from a complete grammar induction system, we also did experiments starting with an automatic clustering of the words in the treebank (details in section 2.1.4. We do not believe that the quality of our tags matches that of the better methods of Schütze (1995), much less the recent results of Clark (2000). Nevertheless, using these tags as input still gave induced structure substantially above right-branching. Figure 5.8 shows the performance with induced tags compared to correct tags. Overall F_1 has dropped, but, interestingly, VP and S recall are higher. This seems to be due to a marked difference between the induced tags and the treebank tags: nouns are scattered among a disproportionately large number of induced tags, increasing the number of common NP sequences, but decreasing the frequency of each.

5.3.4 Convergence and Stability

A common issue with many previous systems is their sensitivity to initial choices. While the model presented here is clearly sensitive to the quality of the input tagging, as well as the qualitative properties of the initial completions, it does not suffer from the need to

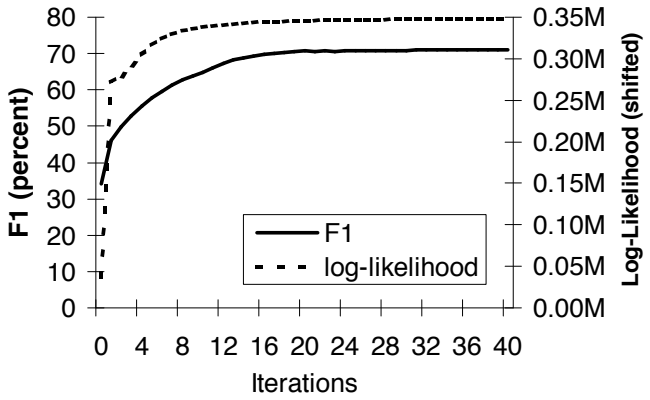


Figure 5.10: F_1 is non-decreasing until convergence.

inject noise to avoid an initial saddle point. Training on random subsets of the training data brought lower performance, but constantly lower over equal-size splits.

Figure 5.10 shows the overall F_1 score and the data likelihood according to our model during convergence.[6] Surprisingly, both are non-decreasing as the system iterates, indicating that data likelihood in this model corresponds well with parse accuracy.[7] Figure 5.12 shows recall for various categories by iteration. NP recall exhibits the more typical pattern of a sharp rise followed by a slow fall, but the other categories, after some initial drops, all increase until convergence.[8] These graphs stop at 40 iterations. The time to convergence varied according to smoothing amount, number of classes, and tags used, but the system almost always converged within 80 iterations, usually within 40.

5.3.5 Partial Supervision

For many practical applications, supplying a few gold parses may not be much more expensive than deploying a fully unsupervised system. To test the effect of partial supervision, we trained the CCM model on 90% of the WSJ10 corpus, and tested it on the remaining

[6]The data likelihood is not shown exactly, but rather we show the linear transformation of it calculated by the system (internal numbers were scaled to avoid underflow).

[7]Pereira and Schabes (1992) find otherwise for PCFGs.

[8]Models in the next chapter also show good correlation between likelihood and evaluation metrics, but generally not monotonic as in the present case.

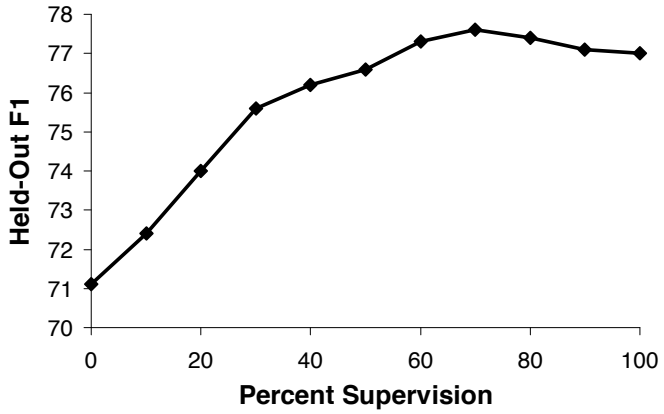


Figure 5.11: Partial supervision

10%. Various fractions of that 90% were labeled with their gold treebank parses; during the learning phase, analyses which crossed the brackets of the labeled parses were given zero weight (but the CCM still filled in binary analyses inside flat gold trees). Figure 5.11 shows F_1 on the held-out 10% as supervision percent increased. Accuracy goes up initially, though it drops slightly at very high supervision levels. The most interesting conclusion from this graph is that small amounts of supervision do not actually seem to help the CCM very much, at least when used in this naive fashion.

5.3.6 Details

There are several details necessary to get good performance out of this model.

Initialization

The completions in this model, just as in the inside-outside algorithm for PCFGs, are distributions over trees. For natural language trees, these distributions are very non-uniform. Figure 5.13 shows empirical bracketing distributions for three languages. These distributions show, over treebank parses of 10-word sentences, the fraction of trees with a constituent over each start and end point. On the other hand, figure 5.14 (b) shows the bracket fractions in a distribution which puts equal weight on each (unlabeled) binary tree. The

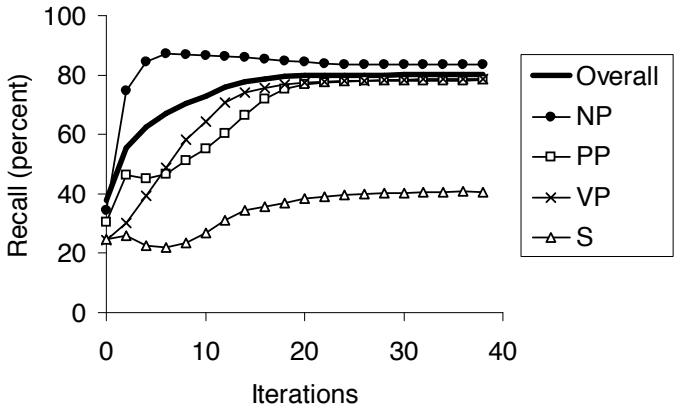
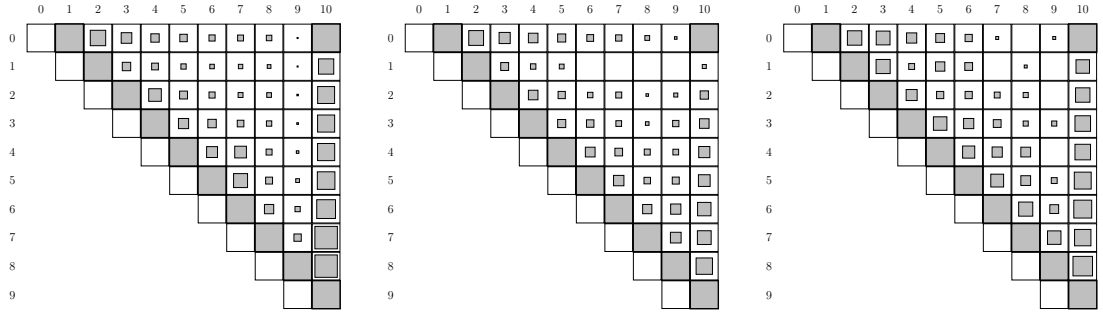


Figure 5.12: Recall by category during convergence.

most important difference between the actual and tree-uniform bracketing distributions is that uniform trees are dramatically more likely to have central constituents, while in natural language constituents tend to either start at the beginning of a sentence or end at the end of the sentence.

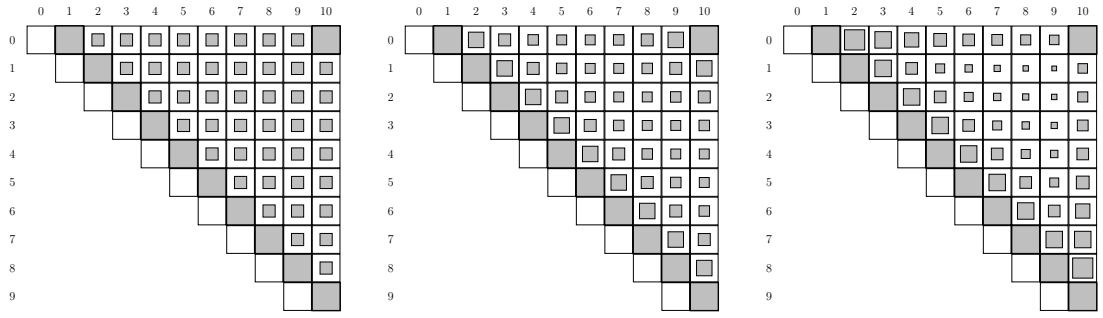
What this means for an induction algorithm is important. Most "uniform" grammars, such as a PCFG in which all rewrites have equal weight, or our current proposal with the constituent and context multinomials being uniform, will have the property that all trees will receive equal scores (or roughly so, modulo any initial perturbation). Therefore, if we begin with an E-step using such a grammar, most first M-steps will be presented with a posterior that looks like figure 5.14(b). If we have a better idea about what the posteriors should look like, we can begin with an E-step instead, such as the one where all non-trivial brackets are equally likely, shown in figure 5.14(a) (this bracket distribution does not correspond to any distribution over binary trees).

Now, we don't necessarily know what the posterior should look like, and we don't want to bias it too much towards any particular language. However, we found that another relatively neutral distribution over trees made a good initializer. In particular, consider the following uniform-splitting process of generating binary trees over k terminals: choose a split point at random, then recursively build trees by this process on each side of the



(a) English (b) German (c) Chinese

Figure 5.13: Empirical bracketing distributions for 10-word sentences in three languages (see chapter 2 for corpus descriptions).



(a) Uniform over Brackets (b) Uniform over Trees (c) Uniform over Splits

Figure 5.14: Bracketing distributions for several notions of "uniform": all brackets having equal likelihood, all trees having equal likelihood, and all recursive splits having equal likelihood.

Initialization	Precision	Recall	F_1	CB
Tree Uniform	55.5	70.5	62.1	1.58
Bracket Uniform	55.6	70.6	62.2	1.57
Split Uniform	64.7	82.2	72.4	0.99
Empirical	65.5	83.2	73.3	1.00

Figure 5.15: CCM performance on WSJ10 as the initializer is varied. Unlike other numbers in this chapter, these values are micro-averaged at the bracket level, as is typical for supervised evaluation, and give credit for the whole-sentence bracket).

split. This process gives a distribution $\mathrm{P}_{\mathrm{split}}$ which puts relatively more weight on unbalanced trees, but only in a very general, non language-specific way. The posterior of the split-uniform distribution is shown in figure 5.14 (c). Another useful property of the split distribution is that it can be calculated in closed form (details in appendix B.2).

In figure 5.13, aside from the well-known right-branching tendency of English (and Chinese), a salient characteristic of all three languages is that central brackets are relatively rare. The split-uniform distribution also shows this property, while the bracket-uniform distribution and the "natural" tree-uniform distribution do not. Unsurprisingly, results when initializing with the bracket-uniform and tree-uniform distributions were substantially worse than using the split-uniform one. Using the actual posterior was, interestingly, only slightly better (see figure 5.15).

While the split distribution was used as an initial completion, it was not used in the model itself. It seemed to bias too strongly against balanced structures, and led to entirely linear-branching structures.

Smoothing

The smoothing used was straightforward, but very important. For each yield α or context x, we added 10 counts of that item: 2 as a constituent and 8 as a distituent. This reflected the relative skew of random spans being more likely to be distituents.

Sentence Length

A weakness of the current model is that it performs much better on short sentences than longer ones: F_1 drops all the way to 53.4% on sentences of length up to 15 (see figure 6.9 in section 6.3). One likely cause is that as spans get longer, span type counts get smaller, and so the parsing is driven by the less-informative context multinomials. Indeed, the primary strength of this system is that it chunks simple NP and PP groups well; longer sentences are less well-modeled by linear spans and have more complex constructions: relative clauses, coordination structures, and so on. The improved models in chapter 6 degrade substantially less with increased sentence length (section 6.3).

5.4 Conclusions

We have presented a simple generative model for the unsupervised distributional induction of hierarchical linguistic structure. The system achieves the above-baseline unsupervised parsing scores on the WSJ10 and ATIS data sets. The induction algorithm combines the benefits of EM-based parameter search and distributional clustering methods. We have shown that this method acquires a substantial amount of correct structure, to the point that the most frequent discrepancies between the induced trees and the treebank gold standard are systematic alternate analyses, many of which are linguistically plausible. We have shown that the system is not overly reliant on supervised POS tag input, and demonstrated increased accuracy, speed, simplicity, and stability compared to previous systems.

Chapter 6

Dependency Models

6.1 Unsupervised Dependency Parsing

Most recent work (and progress) in unsupervised parsing has come from tree or phrase-structure based models, but there are compelling reasons to reconsider unsupervised *dependency* parsing as well. First, most state-of-the-art *supervised* parsers make use of specific lexical information in addition to word-class level information – perhaps lexical information could be a useful source of information for unsupervised methods. Second, a central motivation for using tree structures in computational linguistics is to enable the extraction of dependencies – function-argument and modification structures – and it might be more advantageous to induce such structures directly. Third, as we show below, for languages such as Chinese, which have few function words, and for which the definition of lexical categories is much less clear, dependency structures may be easier to detect.

6.1.1 Representation and Evaluation

An example dependency representation of a short sentence is shown in figure 6.1(a), where, following the traditional dependency grammar notation, the regent or head of a dependency is marked with the tail of the dependency arrow, and the dependent is marked with the arrowhead (Mel′čuk 1988). It will be important in what follows to see that such a representation is isomorphic (in terms of strong generative capacity) to a restricted form of phrase

structure grammar, where the set of terminals and nonterminals is identical, and every rule is of the form X → X Y or X → Y X (Miller 1999), giving the isomorphic representation of figure 6.1(a) shown in figure 6.1(b).[1] Depending on the model, part-of-speech categories may be included in the dependency representation, as suggested here, or dependencies may be directly between words (bilexical dependencies). Below, we will assume an additional reserved nonterminal ROOT, whose sole dependent is the head of the sentence. This simplifies the notation, math, and the evaluation metric.

A dependency analysis will always consist of exactly as many dependencies as there are words in the sentence. For example, in the dependency structure of figure 6.1(b), the dependencies are {(ROOT, *fell*), (*fell, payrolls*), (*fell, in*), (*in, September*), (*payrolls, Factory*)}. The quality of a hypothesized dependency structure can hence be evaluated by accuracy as compared to a gold-standard dependency structure, by reporting the percentage of dependencies shared between the two analyses.

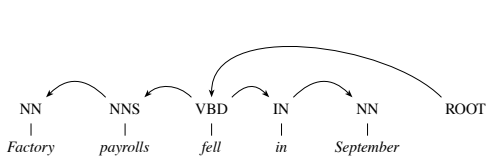
It is important to note that the Penn treebanks do *not* include dependency annotations; however, the automatic dependency rules from (Collins 1999) are sufficiently accurate to be a good benchmark for unsupervised systems for the time being (though see below for specific issues). Similar head-finding rules were used for Chinese experiments. The NEGRA corpus, however, does supply hand-annotated dependency structures.

Where possible, we report an accuracy figure for both directed and undirected dependencies. Reporting undirected numbers has two advantages: first, it facilitates comparison with earlier work, and, more importantly, it allows one to partially obscure the effects of alternate analyses, such as the systematic choice between a modal and a main verb for the head of a sentence (in either case, the two verbs would be linked, but the direction would vary).

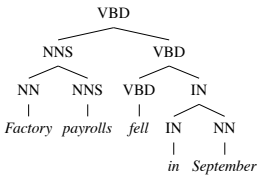
6.1.2 Dependency Models

The dependency induction task has received relatively little attention; the best known work is Carroll and Charniak (1992), Yuret (1998), and Paskin (2002). All systems that we are

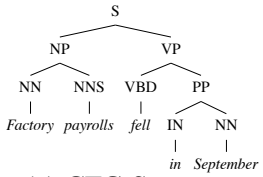
[1] Strictly, such phrase structure trees are isomorphic not to flat dependency structures, but to specific derivations of those structures which specify orders of attachment among multiple dependents which share a common head.



(a) Classical Dependency Structure



(b) Dependency Structure as CF Tree



(c) CFG Structure

Figure 6.1: Three kinds of parse structures.

Figure 6.2: Dependency graph with skeleton chosen, but words not populated.

aware of operate under the assumption that the probability of a dependency structure is the product of the scores of the dependencies (attachments) in that structure. Dependencies are seen as ordered (head, dependent) pairs of words, but the score of a dependency can optionally condition on other characteristics of the structure, most often the direction of the dependency (whether the arrow points left or right).

Some notation before we present specific models: a dependency d is a pair $\langle h, a \rangle$ of a head and an argument, which are words in a sentence s, in a corpus S. For uniformity of notation with chapter 5, words in s are specified as size-one spans of s: for example the first word would be ${}_0s_1$. A dependency structure D over a sentence is a set of dependencies (arcs) which form a planar, acyclic graph rooted at the special symbol ROOT, and in which each word in s appears as an argument exactly once. For a dependency structure D, there is an associated graph G which represents the number of words and arrows between them, without specifying the words themselves (see figure 6.2). A graph G and sentence s together thus determine a dependency structure. The dependency structure is the object generated by all of the models that follow; the steps in the derivations vary from model to

model.

Existing generative dependency models intended for unsupervised learning have chosen to first generate a word-free graph G, then populate the sentence s conditioned on G. For instance, the model of Paskin (2002), which is broadly similar to the semi-probabilistic model in Yuret (1998), first chooses a graph G uniformly at random (such as figure 6.2), then fills in the words, starting with a fixed root symbol (assumed to be at the rightmost end), and working down G until an entire dependency structure D is filled in (figure 6.1a). The corresponding probabilistic model is

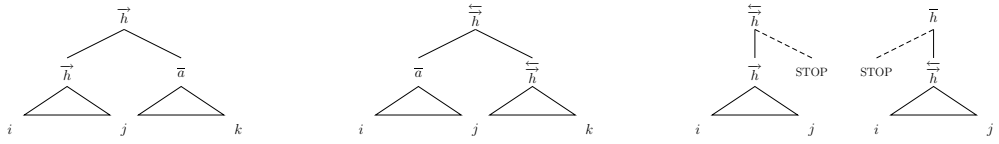
$$\begin{aligned} \mathrm{P}(D) &= \mathrm{P}(s, G) \\ &= \mathrm{P}(G)\mathrm{P}(s|G) \\ &= \mathrm{P}(G) \prod_{(i,j,dir)\in G} \mathrm{P}({}_{i-1}s_i|_{j-1}s_j, dir)\,. \end{aligned}$$

In Paskin (2002), the distribution $\mathrm{P}(G)$ is fixed to be uniform, so the only model parameters are the conditional multinomial distributions $\mathrm{P}(a|h, dir)$ that encode which head words take which other words as arguments. The parameters for left and right arguments of a single head are completely independent, while the parameters for first and subsequent arguments in the same direction are identified.

In those experiments, the model above was trained on over 30M words of raw newswire, using EM in an entirely unsupervised fashion, and at great computational cost. However, as shown in figure 6.3, the resulting parser predicted dependencies at below chance level (measured by choosing a random dependency structure). This below-random performance seems to be because the model links word pairs which have high mutual information (such as occurrences of *congress* and *bill*) regardless of whether they are plausibly syntactically related. In practice, high mutual information between words is often stronger between two topically similar nouns than between, say, a preposition and its object (worse, it's also usually stronger between a verb and a selected preposition than that preposition and its object).

Model	Dir.	Undir.
English (WSJ)		
Paskin 01		39.7
RANDOM		41.7
Charniak and Carroll 92-inspired		44.7
ADJACENT		53.2
DMV		54.4
English (WSJ10)		
RANDOM	30.1	45.6
ADJACENT	33.6	56.7
DMV	43.2	63.7
German (NEGRA10)		
RANDOM	21.8	41.5
ADJACENT	32.6	51.2
DMV	36.3	55.8
Chinese (CTB10)		
RANDOM	35.9	47.3
ADJACENT	30.2	47.3
DMV	42.5	54.2

Figure 6.3: Parsing performance (directed and undirected dependency accuracy) of various dependency models on various treebanks, along with baselines.



(a) (b) (c) (d)

Figure 6.4: Dependency configurations in a lexicalized tree: (a) right attachment, (b) left attachment, (c) right stop, (d) left stop. h and a are head and argument words, respectively, while i, j, and k are positions between words. Not show is the step (if modeled) where the head chooses to generate right arguments before left ones, or the configurations if left arguments are to be generated first.

The specific connection which argues why this model roughly learns to maximize mutual information is that in maximizing

$$\mathrm{P}(D) \;=\; \mathrm{P}(G) \prod_{(i,j,dir)\in G} \mathrm{P}({}_{i-1}s_i|_{j-1}s_j, dir)$$

it is also maximizing

$$\frac{\mathrm{P}(D)}{\mathrm{P}'(D)} \;=\; \frac{\mathrm{P}(G)\prod_{(i,j,dir)\in G}\mathrm{P}({}_{i-1}s_i|_{j-1}s_j, dir)}{\mathrm{P}(G)\prod_i \mathrm{P}({}_{i-1}s_i)}$$

which, dropping the dependence on directionality, gives

$$\begin{aligned}\frac{\mathrm{P}(D)}{\mathrm{P}'(D)} &= \frac{\mathrm{P}(G)\prod_{(i,j)\in G}\mathrm{P}({}_{i-1}s_i|_{j-1}s_j)}{\mathrm{P}(G)\prod_i \mathrm{P}({}_{i-1}s_i)} \\ &= \prod_{(i,j)\in G} \frac{\mathrm{P}({}_{i-1}s_{i,j-1}\,s_j)}{\mathrm{P}({}_{i-1}s_i)\mathrm{P}({}_{j-1}s_j)}\end{aligned}$$

which is a product of (pointwise) mutual information terms.

One might hope that the problem with this model is that the actual lexical items are too semantically charged to represent workable units of syntactic structure. If one were to apply the Paskin (2002) model to dependency structures parameterized simply on the word-classes, the result would be isomorphic to the "dependency PCFG" models described in Carroll and Charniak (1992) (see section 5.1). In these models, Carroll and Charniak considered PCFGs with precisely the productions (discussed above) that make them isomorphic to dependency grammars, with the terminal alphabet being simply parts-of-speech. Here, the rule probabilities are equivalent to $\mathrm{P}(\mathrm{Y}|\mathrm{X}, \textit{right})$ and $\mathrm{P}(\mathrm{Y}|\mathrm{X}, \textit{left})$ respectively.[2] The actual experiments in Carroll and Charniak (1992) do not report accuracies that we can compare to, but they suggest that the learned grammars were of extremely poor quality. As discussed earlier, a main issue in their experiments was that they randomly initialized the production (attachment) probabilities. As a result, their learned grammars were of very

[2]There is another, more subtle distinction: in the Paskin work, a canonical ordering of multiple attachments was fixed, while in the Carroll and Charniak work all attachment orders are considered to be different (equal scoring) structures when listing analyses, giving a relative bias in the Carroll and Charniak work towards structures where heads take more than one argument.

poor quality and had high variance. However, one nice property of their structural constraint, which all dependency models share, is that the symbols in the grammar are not symmetric. Even with a grammar in which the productions are initially uniform, a symbol X can only possibly have non-zero posterior likelihood over spans which contain a matching terminal X. Therefore, one can start with uniform rewrites and let the interaction between the data and the model structure break the initial symmetry. If one recasts their experiments in this way, they achieve an accuracy of 44.7% on the Penn treebank, which is higher than choosing a random dependency structure, but lower than simply linking all adjacent words into a left-headed (and right-branching) structure (53.2%). That this should outperform the bilexical model is in retrospect unsurprising: a major source of non-syntactic information has been hidden from the model, and accordingly there is one fewer unwanted trend that might be detected in the process of maximizing data likelihood.

A huge limitation of both of the above models, however, is that they are incapable of encoding even first-order valence facts, valence here referring in a broad way to the regularities in number and type of arguments a word or word class takes (i.e., including but not limited to subcategorization effects). For example, the former model will attach all occurrences of "new" to "york," even if they are not adjacent, and the latter model learns that nouns to the left of the verb (usually subjects) attach to the verb. But then, given a NOUN NOUN VERB sequence, both nouns will attach to the verb – there is no way that the model can learn that verbs have exactly one subject. We now turn to an improved dependency model that addresses this problem.

6.2 An Improved Dependency Model

The dependency models discussed above are distinct from dependency models used inside high-performance supervised probabilistic parsers in several ways. First, in supervised models, a head outward process is modeled (Eisner 1996, Collins 1999). In such processes, heads generate a sequence of arguments outward to the left or right, conditioning on not only the identity of the head and direction of the attachment, but also on some notion of distance or valence. Moreover, in a head-outward model, it is natural to model stop steps, where the final argument on each side of a head is always the special symbol STOP. Models

like Paskin (2002) avoid modeling STOP by generating the graph skeleton G first, uniformly at random, then populating the words of s conditioned on G. Previous work (Collins 1999) has stressed the importance of including termination probabilities, which allows the graph structure to be generated jointly with the terminal words, precisely because it does allow the modeling of required dependents.

We propose a simple head-outward dependency model over word classes which includes a model of valence, which we call *DMV* (for *dependency model with valence*). We begin at the ROOT. In the standard way (see below), each head generates a series of non-STOP arguments to one side, then a STOP argument to that side, then non-STOP arguments to the other side, then a second STOP.

For example, in the dependency structure in figure 6.1, we first generate a single child of ROOT, here *fell*. Then we recurse to the subtree under *fell*. This subtree begins with generating the right argument *in*. We then recurse to the subtree under *in* (generating *September* to the right, a right STOP, and a left STOP). Since there are no more right arguments after *in*, its right STOP is generated, and the process moves on to the left arguments of *fell*.

In this process, there are two kinds of derivation events, whose local probability factors constitute the model's parameters. First, there is the decision at any point whether to terminate (generate STOP) or not: $\mathrm{P}_{\text{STOP}}(\text{STOP}|h, dir, adj)$. This is a binary decision conditioned on three things: the head h, the direction (generating to the left or right of the head), and the adjacency (whether or not an argument has been generated yet in the current direction, a binary variable). The stopping decision is estimated directly, with no smoothing. If a stop is generated, no more arguments are generated for the current head to the current side. If the current head's argument generation does not stop, another argument is chosen using: $\mathrm{P}_{\text{CHOOSE}}(a|h, dir)$. Here, the argument is picked conditionally on the identity of the head (which, recall, is a word class) and the direction. This term, also, is not smoothed in any way. Adjacency has no effect on the identity of the argument, only on the likelihood of termination. After an argument is generated, its subtree in the dependency structure is recursively generated.

This process should be compared to what is generally done in supervised parsers (Collins 1999, Charniak 2000, Klein and Manning 2003). The largest difference is that supervised parsers condition actions on the identity of the head word itself. The lexical identity is a

good feature to have around in a supervised system, where syntactic lexical facts can be learned effectively. In our unsupervised experiments, having lexical items in the model led to distant topical associations being preferentially modeled over class-level syntactic patterns, though it would clearly be advantageous to discover a mechanism for acquiring the richer kinds of models used in the supervised case. Supervised parsers' decisions to stop or continue generating arguments are also typically conditioned on finer notions of distance than adjacent/non-adjacent (buckets or punctuation-defined distance). Moreover, decisions about argument identity are conditioned on the identity of previous arguments, not just a binary indicator of whether there were any previous ones. This richer history allows for the explicit modeling of inter-argument correlations, such as subcategorization/selection preferences and argument ordering trends. Again, for the unsupervised case, this much freedom can be dangerous. We did not use such richer histories, their success in supervised systems suggests that they could be exploited here, perhaps in a system which originally ignored richer context, then gradually began to model it.

Formally, for a dependency structure D, let each word h have left dependents $deps_D(h, l)$ and right dependents $deps_D(h, r)$. The following recursion defines the probability of the fragment $D(h)$ of the dependency tree rooted at h:

$$\begin{aligned} \mathrm{P}(D(h)) = & \prod_{dir \in \{l,r\}} \prod_{a \in deps_D(h,dir)} \mathrm{P}_{\text{STOP}}(\neg\text{STOP}|h, dir, adj) \\ & \mathrm{P}_{\text{CHOOSE}}(a|h, dir)\mathrm{P}(D(a)) \\ & \mathrm{P}_{\text{STOP}}(\text{STOP}|h, dir, adj) \end{aligned}$$

One can view a structure generated by this derivational process as a "lexicalized" tree composed of the local binary and unary context-free configurations shown in figure 6.4.[3] Each configuration equivalently represents either a head-outward derivation step or a context-free rewrite rule. There are four such configurations. Figure 6.4(a) shows a head h taking a right argument a. The tree headed by h contains h itself, possibly some right

[3]It is lexicalized in the sense that the labels in the tree are derived from terminal symbols, but in our experiments the terminals were word classes, not individual lexical items.

arguments of h, but no left arguments of h (they attach after all the right arguments). The tree headed by a contains a itself, along with all of its left and right children. Figure 6.4(b) shows a head h taking a left argument a – the tree headed by h must have already generated its right stop to do so. Figure 6.4(c) and figure 6.4(d) show the *sealing* operations, where STOP derivation steps are generated. The left and right marks on node labels represent left and right STOPs that have been generated.[4]

The basic inside-outside algorithm (Baker 1979) can be used for re-estimation. For each sentence $s \in S$, it gives us $c_s(x : i, j)$, the expected fraction of parses of s with a node labeled x extending from position i to position j. The model can be re-estimated from these counts. For example, to re-estimate an entry of $\mathrm{P}_{\text{STOP}}(\text{STOP}|h, \textit{left}, \textit{non-adj})$ according to a current model Θ, we calculate two quantities.[5] The first is the (expected) number of trees headed by $\overleftrightarrow{h}$ whose start position i is strictly left of h. The second is the number of trees headed by $\overline{h}$ with start position i strictly left of h. The ratio is the MLE of that local probability factor:

$$\mathrm{P}_{\text{STOP}}(\text{STOP}|h, \textit{left}, \textit{non-adj}) = \frac{\sum_{s \in S} \sum_{i < loc(h)} \sum_k c(\overline{h} : i, k)}{\sum_{s \in S} \sum_{i < loc(h)} \sum_k c(\overleftrightarrow{h} : i, k)}$$

This can be intuitively thought of as the relative number of times a tree headed by h had already taken at least one argument to the left, had an opportunity to take another, but didn't.[6] Section A.2 has a more detailed exposition of the details of calculating the necessary expectations.

Initialization is important to the success of any local search procedure. As in chapter 5, we chose to initialize EM not with an initial model, but with an initial guess at posterior distributions over dependency structures (completions). For the first-round, we constructed

[4]Note that the asymmetry of the attachment rules enforces the right-before-left attachment convention. This is harmless and arbitrary as far as dependency evaluations go, but imposes an X-bar-like structure on the constituency assertions made by this model. This bias/constraint is dealt with in section 6.3.

[5]To simplify notation, we assume each word h occurs at most one time in a given sentence, between indexes $loc(h)$ and $loc(h) + 1$.

[6]As a final note, in addition to enforcing the right-argument-first convention, we constrained ROOT to have at most a single dependent, by a similar device.

a somewhat ad-hoc "harmonic" completion where all non-ROOT words took the same number of arguments, and each took other words as arguments in inverse proportion to (a constant plus) the distance between them. The ROOT always had a single argument and took each word with equal probability. This structure had two advantages: first, when testing multiple models, it is easier to start them all off in a common way by beginning with an M-step, and, second, it allowed us to point the model in the vague general direction of what linguistic dependency structures should look like. It should be emphasized that this initialization was important in getting reasonable patterns out of this model.

On the WSJ10 corpus, the DMV model recovers a substantial fraction of the broad dependency trends: 43.2% of guessed directed dependencies were correct (63.7% ignoring direction). To our knowledge, this is the first published result to break the adjacent-word heuristic (at 33.6% for this corpus). Verbs are the sentence heads, prepositions take following noun phrases as arguments, adverbs attach to verbs, and so on. Figure 6.5 shows the most frequent discrepancies between the test dependencies and the model's guesses. Most of the top mismatches stem from the model systematically choosing determiners to be the heads of noun phrases, where the test trees have the rightmost noun as the head. The model's choice is supported by a good deal of linguistic research (Abney 1987), and is sufficiently systematic that we also report the scores where the NP headship rule is changed to percolate determiners when present. On this adjusted metric, the score jumps hugely to 55.7% directed (and 67.9% undirected). There are other discrepancy types, such as modals dominating main verbs, choice of the wrong noun as the head of a noun cluster, and having some sentences headed by conjunctions.

This model also works on German and Chinese at above-baseline levels (55.8% and 54.2% undirected, respectively), with no modifications whatsoever. In German, the largest source of errors is also the systematic postulation of determiner-headed noun-phrases. The second largest source is that adjectives are (incorrectly) considered to be the head in adjective-noun units. The third source is the incorrect attachment of adjectives into determiners inside definite NPs. Mistakes involving verbs and other classes are less common, but include choosing past participles rather than auxiliaries as the head of periphrastic verb constructions. In Chinese, there is even more severe confusion inside nominal sequences, possibly because the lack of functional syntax makes the boundaries between adjacent NPs

English using DMV			
Overproposals		Underproposals	
DT ← NN	3083	DT → NN	3079
NNP ← NNP	2108	NNP → NNP	1899
CC → ROOT	1003	IN ← NN	779
IN ← DT	858	DT → NNS	703
DT ← NNS	707	NN → VBZ	688
MD → VB	654	NN ← IN	669
DT → IN	567	MD ← VB	657
DT → VBD	553	NN → VBD	582
TO → VB	537	VBD ← NN	550
DT → VBZ	497	VBZ ← NN	543

English using CCM+DMV			
Overproposals		Underproposals	
DT ← NN	3474	DT → NN	3079
NNP ← NNP	2096	NNP → NNP	1898
CD → CD	760	IN ← NN	838
IN ← DT	753	NN → VBZ	714
DT ← NNS	696	DT → NNS	672
DT → IN	627	NN ← IN	669
DT → VBD	470	CD ← CD	638
DT → VBZ	420	NN → VBD	600
NNP → ROOT	362	VBZ ← NN	553
NNS → IN	347	VBD ← NN	528

Figure 6.5: Dependency types most frequently overproposed and underproposed for English, with the DMV alone and with the combination model.

unclear. For example, temporal nouns often take adjacent proper nouns as arguments – all other classes of errors are much less salient.

This dependency induction model is reasonably successful. However, the model can be substantially improved by paying more attention to syntactic constituency, at the same time as modeling dependency structure. To this end, we next present a combined model that exploits kinds of structure. As we will see, the combined model finds correct dependencies more successfully than the model above, and finds constituents more successfully than the model of chapter 5.

6.3 A Combined Model

The CCM and the DMV models have a common ground. Both can be seen as models over lexicalized trees composed of the configurations in figure 6.4. For the DMV, it is already a model over these structures. At the "attachment" rewrite for the CCM in (a/b), we assign the quantity:

$$\frac{\mathrm{P}({}_i s_k|\textit{true})\mathrm{P}({}_{i-1}s_i \sim {}_k s_{k+1}|\textit{true})}{\mathrm{P}({}_i s_k|\textit{false})\mathrm{P}({}_{i-1}s_i \sim {}_k s_{k+1}|\textit{false})}$$

which is the odds ratio of generating the subsequence and context for span $\langle i, k\rangle$ as a constituent as opposed to as a non-constituent. If we multiply all trees' attachment scores by

$$\prod_{\langle i,j\rangle} \mathrm{P}({}_i s_j|\textit{false})\mathrm{P}({}_{i-1}s_i \sim {}_j s_{j+1}|\textit{false})$$

the denominators of the odds ratios cancel, and we are left with each tree being assigned the probability it would have received under the CCM.[7]

In this way, both models can be seen as generating either constituency or dependency structures. Of course, the CCM will generate fairly random dependency structures (constrained only by bracketings). Getting constituency structures from the DMV is also problematic, because the choice of which side to first attach arguments on has ramifications on constituency – it forces x-bar-like structures – even though it is an arbitrary convention as

[7]This scoring function as described is not a generative model over lexicalized trees, because it has no generation step at which nodes' lexical heads are chosen. This can be corrected by multiplying in a "head choice" factor of $1/(k-j)$ at each final "sealing" configuration (d). In practice, this correction factor was harmful for the model combination, since it duplicated a strength of the dependency model, badly.

far as dependency evaluations are concerned. For example, if we attach right arguments first, then a verb with a left subject and a right object will attach the object first, giving traditional VPs, while the other attachment order gives subject-verb groups. To avoid this bias, we alter the DMV in the following ways. When using the dependency model alone, we allow each word to have even probability for either generation order, this order being chosen as the first step in a head's outward dependency generation process (in each actual head derivation, only one order occurs). When using the models together, better performance was obtained by releasing the one-side-attaching-first requirement entirely.[8]

In figure 6.6, we give the behavior of the CCM constituency model and the DMV dependency model on both constituency and dependency induction. Unsurprisingly, their strengths are complementary. The CCM is better at recovering constituency (except for Chinese where neither is working particularly well), and the dependency model is better at recovering dependency structures. It is reasonable to hope that a combination model might exhibit the best of both. In the supervised parsing domain, for example, scoring a lexicalized tree with the product of a simple lexical dependency model and a PCFG model can outperform each factor on its respective metric (Klein and Manning 2003).

In the combined model, we score each tree with the product of the probabilities from the individual models above. We use the inside-outside algorithm to sum over all lexicalized trees, similarly to the situation in section 6.2. The tree configurations are shown in figure 6.4. For each configuration, the relevant scores from each model are multiplied together. For example, consider figure 6.4(a). From the CCM we generate ${}_i s_k$ as a constituent and its corresponding context. From the dependency model, we pay the cost of h taking a as a right argument ($\mathrm{P}_{\mathrm{CHOOSE}}$), as well as the cost of not stopping ($\mathrm{P}_{\mathrm{STOP}}$). The other configurations are similar. We then run the inside-outside algorithm over this product model. From the results, we can extract the statistics needed to re-estimate both individual models.[9]

The models in combination were initialized in the same way as when they were run individually. Sufficient statistics were separately taken off these individual completions. From then on, the resulting models were used together during re-estimation.

[8] With no one-side-first constraint, the proper derivation process chooses whether to stop entirely before each dependent, and if not choose a side to generate on, then generate an argument to that side.

[9] The product, like the CCM itself, is mass-deficient.

	Constituency			Dependency	
Model	UP	UR	UF_1	Dir	Undir
English (WSJ10 – 7422 Sentences)					
LBRANCH/RHEAD	25.6	32.6	28.7	33.6	56.7
RANDOM	31.0	39.4	34.7	30.1	45.6
RBRANCH/LHEAD	55.1	70.0	61.7	24.0	55.9
DMV	46.6	59.2	52.1	43.2	62.7
CCM	64.2	81.6	71.9	23.8	43.3
DMV+CCM (POS)	**69.3**	**88.0**	**77.6**	**47.5**	**64.5**
DMV+CCM (DISTR.)	65.2	82.8	72.9	42.3	60.4
UBOUND	78.8	100.0	88.1	100.0	100.0
German (NEGRA10 – 2175 Sentences)					
LBRANCH/RHEAD	27.4	48.8	35.1	32.6	51.2
RANDOM	27.9	49.6	35.7	21.8	41.5
RBRANCH/LHEAD	33.8	60.1	43.3	21.0	49.9
DMV	38.4	69.5	49.5	40.0	57.8
CCM	48.1	85.5	61.6	25.5	44.9
DMV+CCM	**49.6**	**89.7**	**63.9**	**50.6**	**64.7**
UBOUND	56.3	100.0	72.1	100.0	100.0
Chinese (CTB10 – 2437 Sentences)					
LBRANCH/RHEAD	26.3	48.8	34.2	30.2	43.9
RANDOM	27.3	50.7	35.5	35.9	47.3
RBRANCH/LHEAD	29.0	53.9	37.8	14.2	41.5
DMV	**35.9**	**66.7**	**46.7**	42.5	54.2
CCM	34.6	64.3	45.0	23.8	40.5
DMV+CCM	33.3	62.0	43.3	**55.2**	**60.3**
UBOUND	53.9	100.0	70.1	100.0	100.0

Figure 6.6: Parsing performance of the combined model on various treebanks, along with baselines.

Figure 6.6 summarizes the results. The combined model beats the CCM on English F_1: 77.6 vs. 71.9. To give a concrete indication of how the constituency analyses differ with and without the addition of the dependency model, figure 6.7 shows the sequences which were most frequently overproposed and underproposed as constituents, as well as the crossing proposals, which are the overproposals which actually cross a gold bracket. Note that in the combination model, verb groups disappear and adverbs are handled more correctly (this can be seen in both mistake summaries).

Figure 6.6 also shows the combination model's score when using word classes which were induced entirely automatically, using the same induced classes as in chapter 5. These classes show some degradation, e.g. 72.9 F_1, but it is worth noting that these totally unsupervised numbers are better than the performance of the CCM model running off of Penn treebank word classes. Again, if we modify the gold standard so as to make determiners the head of NPs, then this model with distributional tags scores even better with 50.6% on directed and 64.8% on undirected dependency accuracy.

On the German data, the combination again outperforms each factor alone, though while the combination was most helpful at boosting constituency quality for English, for German it provided a larger boost to the dependency structures. Figure 6.8 shows the common mistake sequences for German, with and without the DMV component. The most dramatic improvement is the more consistent use of verb-object VPs instead of subject-verb groups. Note that for the German data, the gold standard is extremely flat. This is why the precision is so low (49.6% in the combination model) despite the rather high recall (89.7%): in fact the crossing bracket rate is extremely low (0.39, cf. 0.68 for the English combination model).

Finally, on the Chinese data, the combination did substantially boost dependency accuracy over either single factor, but actually suffered a small drop in constituency.[10] Overall, the combination is able to combine the individual factors in an effective way.

To point out one final advantage of the combined model over the CCM (though not the DMV), consider figure 6.9, which shows how the accuracy of the combined model degrades with longer maximum sentence length (10 being WSJ10). On the constituency F_1 measure,

[10]This seems to be partially due to the large number of unanalyzed fragments in the Chinese gold standard, which leave a very large fraction of the posited bracketings completely unjudged.

English using CCM					
Overproposals		Underproposals		Crossing	
JJ NN	1022	NNP NNP	183	MD VB	418
NNP NNP	453	TO CD CD	159	RB VB	306
MD VB	418	NNP POS	159	IN NN	192
DT NN	398	NN NNS	140	POS NN	172
RB VB	349	NN NN	101	CD CD IN CD CD	154
JJ NNS	320	CD CD	72	MD RB VB	148
NNP NN	227	IN CD	69	RB VBN	103
RB VBN	198	TO VB	66	CD NNS	80
IN NN	196	RB JJ	63	VNB TO	72
POS NN	172	IN NNP	62	NNP RB	66

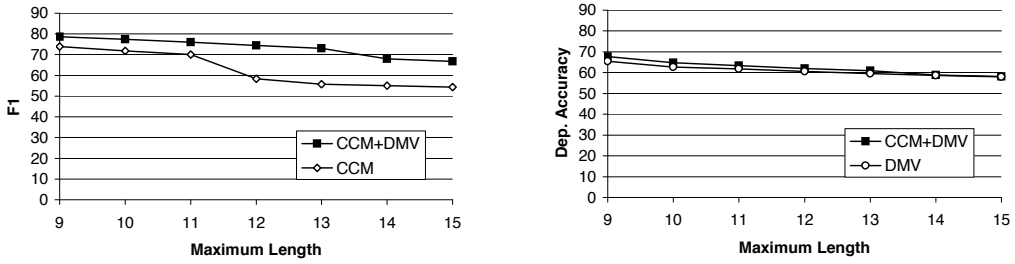
English using CCM+DMV					
Overproposals		Underproposals		Crossing	
JJ NN	1022	NNP NNP	167	CD CD IN CD CD	154
NNP NNP	447	TO CD CD	154	NNS RB	133
DT NN	398	IN NN	76	NNP NNP NNP	67
JJ NNS	294	IN DT NN	65	JJ NN	66
NNP NN	219	IN CD	60	NNP RB	59
NNS RB	164	CD NNS	56	NNP NNP NNP NNP	51
NNP NNP NNP	156	NNP NNP NNP	54	NNP NNP	50
CD CD IN CD CD	155	IN NNP	54	NNS DT NN	41
TO CD CD IN CD CD	154	NN NNS	49	IN PRP	41
CD NN TO CD CD IN CD CD	120	RB JJ	47	RB PRP	33

Figure 6.7: Sequences most frequently overproposed, underproposed, and proposed in locations crossing a gold bracket for English, for the CCM and the combination model.

German using CCM					
Overproposals		Underproposals		Crossing	
ADJA NN	461	APPR ART NN	97	ADJA NN	30
ART NN	430	APPR NN	84	ART NN VVPP	23
ART ADJA NN	94	APPR NE	46	CARD NN	18
KON NN	71	NE NE	32	NN VVPP	16
CARD NN	67	APPR ADJA NN	31	NE NE	15
PPOSAT NN	37	ADV ADJD	24	VVPP VAINF	13
ADJA NN NE	36	ADV ADV	23	ART NN PTKVZ	12
APPRART NN	33	APPR ART ADJA NN	21	NE NN	12
NE NE	30	NN NE	20	NE VVPP	12
ART NN VVPP	29	NN KON NN	19	ART NN VVINF	11

German using CCM+DMV					
Overproposals		Underproposals		Crossing	
ADJA NN	461	NE NE	30	ADJA NN	30
ART NN	430	NN KON NN	22	CARD NN	18
ART ADJA NN	94	NN NE	12	NE NE	17
KON NN	71	APPR NE	9	APPR NN	15
APPR NN	68	ADV PTKNEG	9	ART NN ART NN	14
CARD NN	67	VVPP VAINF	9	NE ADJA NN	11
NE ADJA NN	62	ADV ADJA	9	ADV ADJD	9
NE NE	38	ADV CARD	9	NN APPR NN	8
PPOSAT NN	37	ADJD ADJA	8	ADV NN	7
APPR ART NN	36	CARD CARD	7	APPRART NN NN	7

Figure 6.8: Sequences most frequently overproposed, underproposed, and proposed in locations crossing a gold bracket for German, for the CCM and the combination model.



(a) (b)

Figure 6.9: Change in parse quality as maximum sentence length increases: (a) CCM alone vs. combination and (b) DMV alone vs. combination.

the combination degrades substantially more slowly with sentence length than the CCM alone. This is not too surprising: the CCM's strength is finding common short constituent chunks: the DMV's representation is less scale sensitive at inference time. What is a little surprising is that the DMV and the combination actually converge in dependency accuracy as sentences get longer – this may well be because as sentences get longer, the pressure from the CCM gets relatively weaker: it is essentially agnostic about longer spans.

6.4 Conclusion

We have presented a successful new dependency-based model for the unsupervised induction of syntactic structure, which picks up the key ideas that have made dependency models successful in supervised statistical parsing work. We proceeded to show that it works cross-linguistically. We then demonstrated how this model could be combined with the constituent-induction model of chapter 5 to produce a combination which, in general, substantially outperforms either individual model, on either metric. A key reason that these models are capable of recovering structure more accurately than previous work is that they minimize the amount of hidden structure that must be induced. In particular, neither model attempts to learn intermediate, recursive categories with no direct connection to surface statistics. For example, the CCM models nesting but not recursion. The dependency model is a recursive model, but each tree node is headed by one of the leaves it dominates, so no hidden categories must be posited. Moreover, in their basic forms presented here, neither

model (nor their combination) requires any symmetry-breaking perturbations at initialization.

Chapter 7

Conclusions

There is a great deal of structure in human languages that is not explicit in the observed input. One way or another, human language learners figure out how to analyze, process, generalize, and produce languages to which they are exposed. If we wish to build systems which interact with natural languages, we either have to take a supervised approach and supply detailed annotation which makes all this hidden structure explicit, or we have to develop methods of inducing this structure automatically. The former problem is well-studied and well-understood; the latter problem is merely well-studied. This work has investigated a specific corner of the language learning task: inducing constituency and dependency tree structured analyses given only observations of grammatical sentences (or word-class sequences). We have demonstrated for this task several systems which exceed previous systems' performance in extracting linguistically reasonable structure. Hopefully, we have also provided some useful ideas about what does and does not work for this task, both in our own systems and in other researchers' work.

Many open questions and avenues of research remain, ranging from the extremely technical to the extremely broad. On the narrow side, machine learning issues exist for the present models. Aside from the multi-class CCM, the models have the virtue that symbol symmetries do not arise, so randomness is not needed at initialization. However, all models are sensitive to initialization to some degree. Indeed, for the CCM, one of the contributions of this work is the presentation of a better uniform initializer. In addition, the CCM model family is probabilistically mass deficient; it redundantly generates the observations, and,

most severely, its success may well rely on biases latent in this redundant generation. It performs better on corpora with shorter sentences than longer sentences; part of this issue is certainly that the linear sequences get extremely sparse for long sentences.

A little more broadly, the DMV models, which are motivated by dependency approaches that have performed well in the supervised case, still underperform the theoretically less satisfying CCM models. Despite an enduring feeling that lexical information beyond word-class must be useful for learning language, it seems to be a statistical distraction in some cases. General methods for starting with low-capacity models and gradually releasing model parameters are needed – otherwise we will be stuck with a trade-off between expressivity and learnability that will cap the achievable quality of inducible models.

Much more broadly, an ideal language learning system should not be disconnected from other aspects of language understanding and use, such as the context in which the utterances are situated. Without attempting to learn the meaning of sentences, success at learning their grammatical structure is at best an illuminating stepping stone to other tasks and at worst a data point for linguists interested in nativism. Moreover, from the standpoint of an NLP researcher in need of a parser for a language lacking supervised tools, approaches which are weakly supervised, requiring, say 100 or fewer example parses, are likely to be just as reasonable as fully unsupervised methods, and one could reasonably hope that they would provide better results. In fact, from an engineering standpoint, supplying a few parses is generally much easier than tuning an unsupervised algorithm for a specific language.

Nonetheless, it is important to emphasize that this work has shown that much progress in the unsupervised learning of real, broad-coverage parsers can come from careful understanding of the interaction between the representation of a probabilistic model and what kinds of trends it detects in the data. We can not expect that unsupervised methods will ever exceed supervised methods in cases where there is plenty of labeled training data, but we can hope that, when only unlabeled data is available, unsupervised methods will be important, useful tools, which additionally can shed light on how human languages are structured, used, and learned.

Appendix A

Calculating Expectations for the Models

A.1 Expectations for the CCM

In estimating parameters for the CCM model, the computational bottleneck is the E-step, where we must calculate posterior expectations of various tree configurations according to a fixed parameter vector (chapter 5). This section gives a cubic dynamic program for efficiently collecting these expectations.

The CCM model family is parameterized by two kinds of multinomials: the class-conditional span generation terms $\mathrm{P}_{\mathrm{SPAN}}(\alpha|c)$ and the class-conditional context generation terms $\mathrm{P}_{\mathrm{CONTEXT}}(\beta|c)$, where c is a boolean indicating the constituency of the span, α is the sequence filling that span, and β is the local linear context of that span. The score assigned to a sentence $s_{0,n}$ under a single bracketing B is

$$\mathrm{P}(s, B) = \mathrm{P}_{\mathrm{TREE}}(B) \prod_{\langle i,j \rangle} \mathrm{P}_{\mathrm{SPAN}}(\alpha(i,j,s)|B_{ij}) \mathrm{P}_{\mathrm{CONTEXT}}(\beta(i,j,s)|B_{ij})$$

In $\mathrm{P}_{\mathrm{TREE}}(B)$, the bracketings B with non-zero mass are in one-to-one correspondence with the set of binary tree structures T. Therefore, we can rewrite this expression in terms of the

constituent brackets in the tree $T(B)$.

$$\mathrm{P}(s,B) = \mathrm{P}_{\mathrm{TREE}}(B) \prod_{\langle i,j\rangle \in T(B)} \mathrm{P}_{\mathrm{SPAN}}(\alpha(i,j,s)|\textit{true})\mathrm{P}_{\mathrm{CONTEXT}}(\beta(i,j,s)|\textit{true}) \prod_{\langle i,j\rangle \notin T(B)} \mathrm{P}_{\mathrm{SPAN}}(\alpha(i,j,s)|\textit{false})\mathrm{P}_{\mathrm{CONTEXT}}(\beta(i,j,s)|\textit{false})$$

Since most spans in any given tree are distituents, we can also calculate the score for a bracketing B by starting with the score for the all-distituent bracketing and multiplying in correction factors for the spans which do occur as constituents in B:

$$\mathrm{P}(s,B) = \mathrm{P}_{\mathrm{TREE}}(B) \prod_{\langle i,j\rangle} \mathrm{P}_{\mathrm{SPAN}}(\alpha(i,j,s)|\textit{false})\mathrm{P}_{\mathrm{CONTEXT}}(\beta(i,j,s)|\textit{false}) \prod_{\langle i,j\rangle \in T(B)} \frac{\mathrm{P}_{\mathrm{SPAN}}(\alpha(i,j,s)|\textit{false})\mathrm{P}_{\mathrm{CONTEXT}}(\beta(i,j,s)|\textit{false})}{\mathrm{P}_{\mathrm{SPAN}}(\alpha(i,j,s)|\textit{true})\mathrm{P}_{\mathrm{CONTEXT}}(\beta(i,j,s)|\textit{true})}$$

Since all binary trees have the same score in $\mathrm{P}_{\mathrm{TREE}}$, and the all-distituent product does not depend on the bracketing, we can write this as

$$\mathrm{P}(s,B) = K(s) \prod_{\langle i,j\rangle \in T(B)} \phi(i,j,s)$$

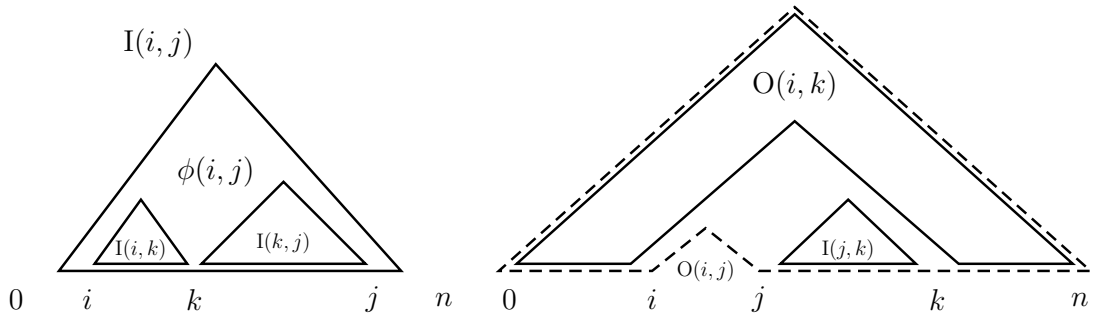
where

$$\phi(i,j,s) = \frac{\mathrm{P}_{\mathrm{SPAN}}(\alpha(i,j,s)|\textit{true})\mathrm{P}_{\mathrm{CONTEXT}}(\beta(i,j,s)|\textit{true})}{\mathrm{P}_{\mathrm{SPAN}}(\alpha(i,j,s)|\textit{false})\mathrm{P}_{\mathrm{CONTEXT}}(\beta(i,j,s)|\textit{false})}$$

and $K(s)$ is some sentence-specific constant. This expression for $\mathrm{P}(s,B)$ now factors according to the nested tree structure of $T(B)$. Therefore, we can define recursions analogous to the standard inside/outside recurrences and use these to calculate the expectations we're interested in.

First, we define $\mathrm{I}(i,j,s)$, which is analogous to the inside score in the inside-outside algorithm for PCFGs.

$$\mathrm{I}(i,j,s) = \sum_{T \in \mathcal{T}(j-i)} \prod_{\langle a,b\rangle : \langle a-i, b-i\rangle \in T} \phi(s,a,b)$$



(a) Inside recurrence

(b) Outside recurrence

Figure A.1: The inside and outside configurational recurrences for the CCM.

In other words, this is the sum, over all binary tree structures T spanning $\langle i, j\rangle$, of the products of the local ϕ scores of the brackets in those trees (see figure A.1). This quantity has a nice recursive decomposition:

$$\mathrm{I}(i,j,s) = \begin{cases} \phi(i,j,s) \sum_{i<k<j} \mathrm{I}(i,k,s)\mathrm{I}(k,j,s) & \text{if } j-i>1 \\ \phi(i,j,s) & \text{if } j-i=1 \\ 0 & \text{if } j-i=0 \end{cases}$$

From this recurrence, either a dynamic programming solution or a memoized solution for calculating the table of I scores in time $O(n^3)$ and space $O(n^2)$ is straightforward.

Similarly, we define $\mathrm{O}(i,j,s)$, which is analogous to the outside score for PCFGs:

$$\mathrm{O}(i,j,s) = \sum_{T \in \mathcal{T}(n-(j-i-1))} \prod_{\langle a,b\rangle \neq \langle i,j\rangle : \langle a,b-(j-i-1)\rangle \in T} \phi(a,b,s)$$

This quantity is the sum of the scores of all tree structures outside the $\langle i, j\rangle$ bracket (again see figure A.1). Note that here, O excludes the factor for the local score at $\langle i, j\rangle$. The

outside sum also decomposes recursively:

$$\mathrm{O}(i,j,s) = \begin{cases} \phi(i,j,s)\sum_{0\leq k<i}\mathrm{I}(k,i,s)\mathrm{O}(k,j,s) + \sum_{j<k\leq n}\mathrm{I}(j,k,s)\mathrm{O}(i,k,s) & \text{if } j-i<n \\ 1 & \text{if } j-i=n \end{cases}$$

Again, the table of O values can be computed by dynamic programming or memoization.

The expectations we need for reestimation of the CCM are the posterior bracket counts $\mathrm{P}_{\mathrm{BRACKET}}(i,j|s)$, the fraction of trees (bracketings) that contain the span $\langle i,j,\rangle$ as a constituent.

$$\mathrm{P}_{\mathrm{BRACKET}}(i,j|s) = \frac{\sum_{B:B(i,j)=\mathit{true}} P(s,B)}{\sum_B P(s,B)}$$

We can calculate the terms in this expression using the I and O quantities. Since the set of trees containing a certain bracket is exactly the cross product of partial trees inside that bracket and partial trees outside that bracket, we have

$$\sum_{B:B(i,j)=\mathit{true}} P(s,B) = K(s)\mathrm{I}(i,j,s)\mathrm{O}(i,j,s)$$

and

$$\sum_B P(s,B) = K(s)\mathrm{I}(0,n,s)\mathrm{O}(0,n,s)$$

Since the constants $K(s)$ cancel and $\mathrm{O}(0,n,s)=1$, we have

$$\mathrm{P}_{\mathrm{BRACKET}}(i,j|s) = \frac{\mathrm{I}(i,j,s)\mathrm{O}(i,j,s)}{\mathrm{I}(0,n,s)}$$

A.2 Expectations for the DMV

The DMV model can be most simply (though not most efficiently) described as decomposing over a lexicalized tree structure, as shown in figure A.2. These trees are essentially context-free trees in which the terminal symbols are $w \in W$ for some terminal vocabulary W (here, W is the set of word-classes). The non-terminal symbols are $\overrightarrow{w}$, $\overleftarrow{w}$, $\overleftrightarrow{w}$, $\overrightarrow{\overleftrightarrow{w}}$, and $\overline{w}$, for $w \in W \cup \{\blacklozenge\}$. The productions are of the following forms:

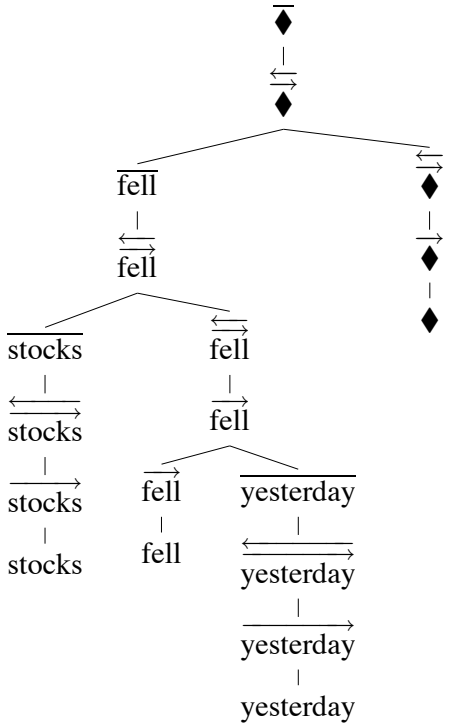


Figure A.2: A lexicalized tree in the fully articulated DMV model.

Head choice (right-first)	$\overrightarrow{w} \rightarrow w$
Head choice (left-first)	$\overleftarrow{w} \rightarrow w$
Right-first right attachment	$\overrightarrow{h} \rightarrow \overrightarrow{h} \ \ \overline{a}$
Right-first right stop	$\overleftrightarrow{h} \rightarrow \overrightarrow{h}$
Right-first left attachment	$\overleftrightarrow{h} \rightarrow \overline{a} \ \ \overleftrightarrow{h}$
Right-first seal	$\overline{h} \rightarrow \overleftrightarrow{h}$
Left-first left attachment	$\overleftarrow{h} \rightarrow \overline{a} \ \ \overleftarrow{h}$
Left-first left stop	$\overrightleftarrow{h} \rightarrow \overleftarrow{h}$
Left-first right attachment	$\overrightleftarrow{h} \rightarrow \overrightleftarrow{h} \ \ \overline{a}$
Left-first seal	$\overline{h} \rightarrow \overrightleftarrow{h}$

We imagine all sentences to end with the symbol ♦, and treat $\overline{\blacklozenge}$ as the start symbol. Trees with root $\overline{h}$ are called *sealed* trees, since their head h cannot take any further arguments in this grammar topology. Trees with roots $\overleftrightarrow{h}$ and $\overrightleftarrow{h}$ are called *half-sealed* trees, since they can only take arguments to the final attachment side (left and right, respectively). Trees rooted at $\overrightarrow{h}$ and $\overleftarrow{h}$ are called *unsealed* since they can take arguments to either side (eventually).

Derivations of a sentence in the DMV dependency model correspond to parses with the above grammar. However, each configuration is scored according to a head-outward derivation (Collins 1999), rather than a top-down rewrite. For example, the right-first head choice is thought of as deciding, given the head, that in this derivation the head will attach its right arguments before its left arguments. This configuration incurs the probability factor $\mathrm{P}_{\mathrm{ORDER}}(\textit{right-first}|w)$. The other configurations indicate attachments or stopping probabilities, as described in section 6.2.

We define inside and outside recurrences over items $(\mathrm{X} : i, j)$ in the standard way. The base cases are projections of terminals, which represent the point where we decide whether a word w will take its arguments to the right then left, or left then right, X-bar style.

$$\mathrm{P}_{\mathrm{INSIDE}}(x, i, i+1) = \begin{cases} \mathrm{P}_{\mathrm{ORDER}}(\textit{left-first}|w) & \text{if } x = \overleftarrow{w} \\ \mathrm{P}_{\mathrm{ORDER}}(\textit{right-first}|w) & \text{if } x = \overrightarrow{w} \end{cases}$$

For spans greater than one, we have to sum over the various decompositions. Sealed scores are expressed in terms of half-sealed scores:

$$\begin{aligned}\mathrm{P}_{\text{INSIDE}}(\overline{w}, i, j) = \mathrm{P}_{\text{STOP}}(\text{stop}|w, \text{left}, \text{adj}(i, w))\mathrm{P}_{\text{INSIDE}}(\overleftarrow{\overrightarrow{w}}, i, j) + \\ \mathrm{P}_{\text{STOP}}(\text{stop}|w, \text{right}, \text{adj}(j, w))\mathrm{P}_{\text{INSIDE}}(\overrightarrow{\overleftarrow{[}} w, i, j)\end{aligned}$$

Half-sealed scores are expressed in terms of either smaller half-sealed scores or unsealed scores:

$$\begin{aligned}\mathrm{P}_{\text{INSIDE}}(\overleftarrow{\overrightarrow{w}}, i, j) = \Bigg(\sum_k \sum_a \mathrm{P}_{\text{STOP}}(\neg\text{stop}|w, \text{left}, \text{adj}(k, w))\mathrm{P}_{\text{ATTACH}}(a|w, \text{left}) \times \\ \mathrm{P}_{\text{INSIDE}}(\overline{a}, i, k)\mathrm{P}_{\text{INSIDE}}(\overleftarrow{\overrightarrow{w}}, k, j)\Bigg) + \\ \mathrm{P}_{\text{STOP}}(\text{stop}|w, \text{right}, \text{adj}(j, w))\mathrm{P}_{\text{INSIDE}}(\overrightarrow{\overleftarrow{w}}, i, j) \\ \mathrm{P}_{\text{INSIDE}}(\overrightarrow{\overleftarrow{w}}, i, j) = \Bigg(\sum_k \sum_a \mathrm{P}_{\text{STOP}}(\neg\text{stop}|w, \text{right}, \text{adj}(k, w))\mathrm{P}_{\text{ATTACH}}(a|w, \text{right}) \times \\ \mathrm{P}_{\text{INSIDE}}(\overrightarrow{\overleftarrow{w}}, i, k)\mathrm{P}_{\text{INSIDE}}(\overline{a}, k, j)\Bigg) + \\ \mathrm{P}_{\text{STOP}}(\text{stop}|w, \text{left}, \text{adj}(i, w))\mathrm{P}_{\text{INSIDE}}(\overleftarrow{w}, i, j)\end{aligned}$$

Note the dependency on adjacency: the function $\text{adj}(i, w)$ indicates whether the index i is adjacent to word w (on either side).[1] Unsealed scores (for spans larger than one) are

[1] Note also that we're abusing notation so that w indicates not just a terminal symbol, but a specific instance of that symbol in the sentence.

expressed in terms of smaller unsealed scores:

$$\mathrm{P}_{\mathrm{INSIDE}}(\overrightarrow{w}, i, j) = \sum_k \sum_a \mathrm{P}_{\mathrm{STOP}}(\neg\mathrm{stop}|w, \mathrm{right}, \mathrm{adj}(k, w))\mathrm{P}_{\mathrm{ATTACH}}(a|w, \mathrm{right}) \times \mathrm{P}_{\mathrm{INSIDE}}(\overrightarrow{w}, i, k)\mathrm{P}_{\mathrm{INSIDE}}(\overline{a}, k, j)$$

$$\mathrm{P}_{\mathrm{INSIDE}}(\overleftarrow{w}, i, j) = \sum_k \sum_a \mathrm{P}_{\mathrm{STOP}}(\neg\mathrm{stop}|w, \mathrm{left}, \mathrm{adj}(k, w))\mathrm{P}_{\mathrm{ATTACH}}(a|w, \mathrm{left}) \times \mathrm{P}_{\mathrm{INSIDE}}(\overline{a}, i, k)\mathrm{P}_{\mathrm{INSIDE}}(\overrightarrow{w}, k, j)$$

The outside recurrences for $\mathrm{P}_{\mathrm{OUTSIDE}}(x, i, j)$ are similar. Both can be calculated in $O(n^5)$ time using the standard inside-outside algorithms for headed (lexicalized) PCFGs or memoization techniques. Of course, the $O(n^4)$ and $O(n^3)$ techniques of Eisner and Satta (1999) apply here, as well, but not in the case of combination with the CCM model, and are slightly more complex to present.

In any case, once we have the inside and outside scores, we can easily calculate the fraction of trees over a given sentence which contain any of the structural configurations which are necessary to re-estimate the model multinomials.

A.3 Expectations for the CCM+DMV Combination

For the combination of the CCM and DMV models, we used the simplest technique which admitted a dynamic programming solution. For any lexicalized derivation tree structure of the form shown in figure A.2, we can read off a list of DMV derivation stops (stops, attachments, etc.). However, we can equally well read off a list of assertions that certain sequences and their contexts are constituents or distituents. Both models therefore assign a score to a lexicalized derivation tree, though multiple distinct derivation trees will contain the same set of constituents.

To combine the models, we took lexicalized derivation trees T and scored them with

$$\mathrm{P}_{\mathrm{COMBO}}(T) = \mathrm{P}_{\mathrm{CCM}}(T)\mathrm{P}_{\mathrm{DMV}}(T)$$

The quantity $\mathrm{P}_{\mathrm{COMBO}}$ is a deficient probability function, in that, even if $\mathrm{P}_{\mathrm{CCM}}$ were not itself deficient, the sum of probabilities over all trees T will be less than one.[2]

Calculating expectations with respect to $\mathrm{P}_{\mathrm{COMBO}}$ is almost exactly the same as working with $\mathrm{P}_{\mathrm{DMV}}$. We use a set of $O(n^5)$ recurrences, as in section A.2. The only difference is that we must premultiply all our probabilities by the CCM base product

$$\prod_{\langle i,j \rangle} \mathrm{P}_{\mathrm{SPAN}}(\alpha(i,j,s)|\textit{false})\mathrm{P}_{\mathrm{CONTEXT}}(\beta(i,j,s)|\textit{false})$$

and we must multiply in the local CCM correction factor $\phi(i,j,s)$ at each constituent that spans more than one terminal. To do the latter, we simply adjust the binary-branching recurrences above. Instead of:

$$\begin{aligned}
\mathrm{P}_{\mathrm{INSIDE}}(\overleftarrow{\overrightarrow{w}}, i, j) = \Bigg(\sum_k \sum_a \mathrm{P}_{\mathrm{STOP}}(\neg\mathrm{stop}|w, \mathrm{left}, \mathrm{adj}(k,w))\mathrm{P}_{\mathrm{ATTACH}}(a|w,\mathrm{left}) \times \\
\mathrm{P}_{\mathrm{INSIDE}}(\overline{a}, i, k)\mathrm{P}_{\mathrm{INSIDE}}(\overleftarrow{\overrightarrow{w}}, k, j) \Bigg) + \\
\mathrm{P}_{\mathrm{STOP}}(\mathrm{stop}|w, \mathrm{right}, \mathrm{adj}(j,w))\mathrm{P}_{\mathrm{INSIDE}}(\overrightarrow{w}, i, j)
\end{aligned}$$

$$\begin{aligned}
\mathrm{P}_{\mathrm{INSIDE}}(\overrightarrow{\overleftarrow{w}}, i, j) = \Bigg(\sum_k \sum_a \mathrm{P}_{\mathrm{STOP}}(\neg\mathrm{stop}|w, \mathrm{right}, \mathrm{adj}(k,w))\mathrm{P}_{\mathrm{ATTACH}}(a|w,\mathrm{right}) \times \\
\mathrm{P}_{\mathrm{INSIDE}}(\overrightarrow{\overleftarrow{w}}, i, k)\mathrm{P}_{\mathrm{INSIDE}}(\overline{a}, k, j) \Bigg) + \\
\mathrm{P}_{\mathrm{STOP}}(\mathrm{stop}|w, \mathrm{left}, \mathrm{adj}(i,w))\mathrm{P}_{\mathrm{INSIDE}}(\overleftarrow{w}, i, j)
\end{aligned}$$

[2]Except in degenerate cases, such as if both components put mass one on a single tree.

we get

$$\mathrm{I}(\overleftarrow{\overrightarrow{w}}, i, j|s) = \Bigg(\sum_k \sum_a \mathrm{P}_{\mathrm{STOP}}(\neg\mathrm{stop}|w, \mathrm{left}, \mathrm{adj}(k, w))\mathrm{P}_{\mathrm{ATTACH}}(a|w, \mathrm{left})\mathrm{P}_{\mathrm{INSIDE}}(\overline{a}, i, k) \times \mathrm{P}_{\mathrm{INSIDE}}(\overleftarrow{\overrightarrow{w}}, k, j)\phi(i, j, s)\Bigg) + \mathrm{P}_{\mathrm{STOP}}(\mathrm{stop}|w, \mathrm{right}, \mathrm{adj}(j, w))\mathrm{P}_{\mathrm{INSIDE}}(\overrightarrow{w}, i, j)$$

$$\mathrm{I}(\overrightarrow{\overleftarrow{w}}, i, j|s) = \Bigg(\sum_k \sum_a \mathrm{P}_{\mathrm{STOP}}(\neg\mathrm{stop}|w, \mathrm{right}, \mathrm{adj}(k, w))\mathrm{P}_{\mathrm{ATTACH}}(a|w, \mathrm{right}) \times \mathrm{P}_{\mathrm{INSIDE}}(\overrightarrow{\overleftarrow{w}}, i, k)\mathrm{P}_{\mathrm{INSIDE}}(\overline{a}, k, j)\phi(i, j, s)\Bigg) + \mathrm{P}_{\mathrm{STOP}}(\mathrm{stop}|w, \mathrm{left}, \mathrm{adj}(i, w))\mathrm{P}_{\mathrm{INSIDE}}(\overleftarrow{w}, i, j)$$

and similarly

$$\mathrm{P}_{\mathrm{INSIDE}}(\overrightarrow{w}, i, j) = \sum_k \sum_a \mathrm{P}_{\mathrm{STOP}}(\neg\mathrm{stop}|w, \mathrm{right}, \mathrm{adj}(k, w))\mathrm{P}_{\mathrm{ATTACH}}(a|w, \mathrm{right}) \times \mathrm{P}_{\mathrm{INSIDE}}(\overrightarrow{w}, i, k)\mathrm{P}_{\mathrm{INSIDE}}(\overline{a}, k, j)$$

$$\mathrm{P}_{\mathrm{INSIDE}}(\overleftarrow{w}, i, j) = \sum_k \sum_a \mathrm{P}_{\mathrm{STOP}}(\neg\mathrm{stop}|w, \mathrm{left}, \mathrm{adj}(k, w))\mathrm{P}_{\mathrm{ATTACH}}(a|w, \mathrm{left}) \times \mathrm{P}_{\mathrm{INSIDE}}(\overline{a}, i, k)\mathrm{P}_{\mathrm{INSIDE}}(\overrightarrow{w}, k, j)$$

becomes

$$\mathrm{I}(\overrightarrow{w}, i, j|s) = \sum_k \sum_a \mathrm{P}_{\mathrm{STOP}}(\neg\mathrm{stop}|w, \mathrm{right}, \mathrm{adj}(k, w))\mathrm{P}_{\mathrm{ATTACH}}(a|w, \mathrm{right}) \times \mathrm{P}_{\mathrm{INSIDE}}(\overrightarrow{w}, i, k)\mathrm{P}_{\mathrm{INSIDE}}(\overline{a}, k, j)\phi(i, j, s)$$

$$\mathrm{I}(\overleftarrow{w}, i, j|s) = \sum_k \sum_a \mathrm{P}_{\mathrm{STOP}}(\neg\mathrm{stop}|w, \mathrm{left}, \mathrm{adj}(k, w))\mathrm{P}_{\mathrm{ATTACH}}(a|w, \mathrm{left}) \times \mathrm{P}_{\mathrm{INSIDE}}(\overline{a}, i, k)\mathrm{P}_{\mathrm{INSIDE}}(\overrightarrow{w}, k, j)\phi(i, j, s)$$

Again, the outside expressions are similar. Notice that the scores which were inside probabilities in the DMV case are now only sum-of-products of scores, relativized to the current sentence, just as in the CCM case.