Top-Down Nearly-Context-Sensitive Parsing

Eugene Charniak
Brown Laboratory for Linguistic Information Processing (BLLIP)
Brown University, Providence, RI 02912
ec@cs.brown.edu

Abstract

We present a new syntactic parser that works left-to-right and top down, thus maintaining a fully-connected parse tree for a few alternative parse hypotheses. All of the commonly used statistical parsers use context-free dynamic programming algorithms and as such work bottom up on the entire sentence. Thus they only find a complete fully connected parse at the very end. In contrast, both subjective and experimental evidence show that people understand a sentence word-to-word as they go along, or close to it. The constraint that the parser keeps one or more fully connected syntactic trees is intended to operationalize this cognitive fact. Our parser achieves a new best result for top-down parsers of 89.4%,a 20% error reduction over the previous single-parser best result for parsers of this type of 86.8% (Roark, 2001). The improved performance is due to embracing the very large feature set available in exchange for giving up dynamic programming.

1 Introduction

We present a new syntactic parser that works top-down and left-to-right, maintaining a fully-connected parse tree for a few alternative parse hypotheses. It is a Penn treebank (Marcus et al., 1993) parser in that it is capable of parsing the Penn treebank test sets, and is trained on the now standard training set. It achieves a new best result for this parser type.

All of the commonly used statistical parsers available on the web such as the Collins(/Bikel) (Collins, 2003) Charniak-Johnson(Charniak and Johnson, 2005), and Petrov-Klein (Petrov et al., 2006), parsers use context-free dynamic programming algorithms so they work bottom up on the entire sentence. Thus they only find a complete fully-connected parse at the very end.

In contrast human syntactic parsing must be fully connected (or close to it) as people are able to apply vast amounts of real-world knowledge to the process as it proceeds from word-to-word(van Gompel and Pickering, 2007). Thus any parser claiming cognitive plausibility must, to a first approximation, work in this left-to-right top-down fashion.

Our parser obtains a new best result for top-down parsers of 89.4% (on section 23 of the Penn Treebank). This is a 20% error reduction over the previous best single-parser result of 86.8%, achieved by Roark(Roark, 2001).

Our model is in the tradition of this latter parser. The current work's superior performance is not due to any innovation in architecture but in how probability distributions are computed. It differs from Roark in its explicit recognition that by giving up context-free dynamic programming we may embrace near context sensitivity and condition on many diverse pieces of information. (It is only "near" because we still only condition on a finite amount of information.) This is made possible by use of random-forests (Amit and Geman, 1997; Breiman, 2004; Xu and Jelinek, 2004) to choose features, provide smoothing, and finally do the probability computation. To the best of our knowledge ours is the first application of random-forests to parsing.

Section two describes previous work on this type of parser, and in particular gives details on the Roark architecture we use. Section three describes how random forests allow us to integrate the diverse information sources that context-sensitive parsing allows. Section four gives implementation details. Section five is devoted to the main experimental finding of the paper along with subsidiary results showing the effects of the large feature set we now may use. Finally, section six suggests that because this parser type is comparatively little explored one may hope for further substantial improvements, and proposes avenues to be explored.

2 Previous Work on Top-Down Parsing and the Roark Model

We care about top-down incremental parsing because it automatically satisfies the criteria we have established for cognitive plausibility. Before looking at previous work on this type model we briefly discuss work that does *not* meet the criteria we have set out, but which people often assume does so. In particular we discuss the work of Nivre(Nivre, 2003) and Henderson(Henderson, 2003). The reason these are not fully-connected is the same. While they are incremental parsers, they are not top down — both are shift-reduce parsers. Consider a constituency shift-reduce mechanism. Suppose we have a context-free rule S → NP VP. As we go left-to-right various terminal and non-terminals are added to and removed from the stack until at some point the top two items are NP and VP. Then a reduce operation replaces them with S. Note that this means that *none* of the words or sub-constituents of either the NP or VP are integrated into a single overall tree until the very end. This is clearly not fully connected. Since Nivre's parser is a dependency parser this exact case does not apply (as it does not use CFG rules), but similar situations arise. In particular, whenever a word is dependent on a word that appears later in the string, it remains unconnected on the stack until the second word appears. Naturally this is transitive so that the parser can, and presumably does, process an unbounded number of words before connecting them all together.

parse $(w_{0,n-1})$
1 $C[0](= h =< q, r, t >) \leftarrow < 1, 1, ROOT >$
2 for $i = 0, n$
3 do while ABOVE-THRESHOLD (h, C, N)
4 remove h from C
5 for all x such that $p(x \mid t) > 0$
6 let $h' =< q', r', t' >$
7 where $q' = q * p(x \mid t)$,
$r' = \text{LAP}(t', w')$,
and $t' = t \circ x$
8 if$(x = w)$ then $w' = w_{i+1}$
insert h' in N
9 else $w' = w$
10 insert h' in C
11 empty C
12 exchange C and N
13 output $t(C[0])$.

Figure 1: Roark's Fully-Connected Parsing Algorithm

Here we follow the work of Roark (Roark, 2001) which *is* fully-connected. The basic algorithm is given in Figure 1. (Note we have simplified the algorithm in several ways.) The input to the parser is a string of n words $w_{0,n-1}$. We pad the end of the string with an end-of-sentence marker $w_n = \triangleleft$. This has the special property that $p(\triangleleft \mid t) = 1$ for a complete tree t of $w_{0,n-1}$, zero otherwise.

There are two priority queues of hypotheses, C (current), and N (next). A hypothesis h is a three-tuple $< q, r, t >$ where q is the probability assigned to the current tree t. In Figure 1 h always denotes $C[0]$ the top-most element of C. While we call t the "tree", it is a vector representation of a tree. For example, the tree (ROOT) would be a vector of two elements, ROOT and ")", the latter indicating that the constituent labeled root is completed. Thus elements of the vector are the terminals, non-terminals, and ")" — the close parenthesis element. Lastly r is the "look-ahead probability" or LAP. LAP(w,h) is (a crude estimate of) the probability that the

	Current hypotheses	Prob of next tree element
1	$< 1.4 * 10^{-3}, 5 * 10^{-2}$,(S (NP (NNS Terms)>	p(x=")" | t)=.64
2	$< 7 * 10^{-5}, 5 * 10^{-2}$,(S (S (NP (NNS Terms)>	
3	$< 9 * 10^{-4}, 8 * 10^{-2}$,(S (NP (NNS Terms))>	p(x=VP | t) =.88
4		p(x=S | t)= $2 * 10^{-4}$
5	$< 7 * 10^{-5}, 5 * 10^{-2}$,(S (S (NP (NNS Terms)>	
6	$< 9 * 10^{-4}, 9 * 10^{-2}$,(S (NP (NNS Terms)) (VP>	p(x=AUX | t)= .38
7	$< 7 * 10^{-5}, 5 * 10^{-2}$,(S (S (NP (NNS Terms)>	
8	$< 2 * 10^{-8}, 9 * 10^{-2}$,(S (NP (NNS Terms)) (S>	
9	$< 3 * 10^{-4}, 2 * 10^{-1}$,(S (NP (NNS Terms)) (VP (AUX >	p(x="were" | t)= .21
10	$< 7 * 10^{-5}, 5 * 10^{-2}$,(S (S (NP (NNS Terms)>	
11	$< 2 * 10^{-8}, 9 * 10^{-2}$,(S (NP (NNS Terms)) (S>	
12	$< 7 * 10^{-5}, 3 * 10^{-1}$,(S (NP (NNS Terms)) (VP (AUX were)>	

Figure 2: Parsing the second word of "Terms were not disclosed."

next word is w given h. We explain its purpose below.

We go through the words one at a time. At the start of our processing of w_i we have hypotheses on C ordered by $p \cdot q$ — the probability of the hypothesis so far times an estimate q of the probability cost we encounter when trying to now integrate w_i. We remove each h from C and integrate a new tree symbol x. If $x = w_i$ it means that we have successfully added the new word to the tree and this hypothesis goes into the queue for the next word N. Otherwise h does not yet represent an extension to w_i and we put it back on C to compete with the other hypotheses waiting to be extended. The look-ahead probability $\mathrm{LAP}(h) = q$ is intended to keep a level playing field. If we put h back onto C it's probability p is lowered by the factor $p(x \mid h)$. On the other hand, if x is the correct symbol, q should go up, so the two should offset and h is still competitive.

We stop processing a word and move onto the next when ABOVE-THRESHOLD returns false. Without going into details, we have adopted exactly the decision criteria and associated parameters used by Roark so that the accuracy numbers presumably reflect the same amount of search. (The more liberal ABOVE-THRESHOLD, the more search, and presumably the more accurate results, everything else being equal.)

Figure 2 shows a few points in the processing of the second word of "Terms were not disclosed." Lines one and two show the current queue at the start of processing. Line one has the ultimately correct partial tree (S (NP (NNS Terms). Note that the NP is not closed off as the parser defers closing constituents until necessary. On the right of line 1 we show the possible next tree pieces that could be added. Here we simply have one, namely a right parenthesis to close off the NP. (In reality there would be many such x's.) The result is that the hypothesis of line 1 is removed from the queue, and a new hypothesis is added back on C as this new hypothesis does not incorporate the second word.

Lines 3 and 5 now show the new state of C. Again we remove the top candidate from C. The right-hand side of lines 3 and 4 show two possible continuations for the h of line 3, start a new VP or a new S. With line 3 removed from the queue, and its two extensions added, we get the new queue state shown in lines 6,7 and 8. Line 6 shows the top-most hypothesis extended by an AUX. This still has not yet incorporated

the next word into the parse, so this extension is inserted in the current queue giving us the queue state shown in 9,10,11. Finally line 9 is extended with the word "were." This addition incorporates the current word, and the resulting extension, shown in line 12 is inserted in N, not C, ending this example.

3 Random Forests

The Roark model we emulate requires the estimation of two probability distributions: one for the next tree element (non-terminals,terminals, and ")") in the grammar, and one for the look-ahead probability of the yet-to-be-incorporated next word. In this section we use the first of these for illustration.

We first consider how to construct a single (non-random) decision tree for estimating this distribution. A tree is a fully-connected directed acyclic graph where each node has one input arc (except for the root) and, for reasons we go into later, either zero or two output arcs — the tree is binary. A node is a four-tuple $< d, s, p, q >$, where d is a set of training instances, p, a probability distribution of the correct decisions for all of the examples in d, and q a binary question about the conditioning information for the examples in d. The 0/1 answer to this question causes the decision-tree program to follow the left/right arc out of the node to the children nodes. If q is null, the node is a leaf node. s is a strict subset of the domain of the q for the parent of h.

Decision tree construction starts with the root node n where d consists of the several million situations in the training data where the next tree element needs to be guessed (according to our probability distribution) based upon previous words and the analysis so far. At each iteration one node is selected from a queue of unprocessed nodes. A question q is selected, and based upon its answers two descendents n_1 and n_2 are created with d_1 and d_2 respectively, $d_1 \cup d_2 = d$. These are inserted in the queue of unprocessed nodes and the process repeats. Termination can be handled in multiple ways. We have chosen to simply pick the number of nodes we create.

No.	Q	S	p("of")	p("in")
0	1			
1	1	NN,IN	0.05	0.03
4	1	NNS,IN	0.09	0.06
12	2	RB,IN	0.17	0.11
16	3	PP,WHPP	0.27	0.18
39	20	NP,NX	0.51	0.16
40	20	S,VP	0.0004	0.19

Figure 3: Some nodes in a decision tree for $p(w_i \mid t)$

Nodes left on the queue are the leaf nodes of the decision tree. We pick nodes from the heap based upon how much they increased the probability of the data.

Still open is the selection of q at each iteration. First pick a query type qt from a user supplied set. In our case there are 27 types. Examples include the parent of the non-terminal to be created, its predecessor, 2 previous, etc. A complete list is given in Figure 4. Note that the answers to these queries are *not* binary.

Secondly we turn our qt into a binary question by creating two disjoint sets s_1 and s_2 $s_1 \cup s_2 = s$ where s is the domain of qt. If a particular history $h \in d$ is such that $qt(h) = x$ and $x \in s_1$ then h is put in d_1. Similarly for s_2. For example, if qt is the parent relation, and the parent in h is NP, then h goes in d_1 iff NP $\in s_1$. We create the sets s_i by initializing them randomly, and then for each $x \in s$ try moving x to the opposite set s_i. If this results in a higher data probability we keep it in its new s_i, otherwise it reverts to it's original s_i. This is repeated until no switch lowers probability. (Or were the a's are individual words, until no more than two words switch.)

We illustrate with a concrete example. One important fact quickly impressed on the creator of parsers is the propensity of prepositional phrases (PP) headed by "of" to attach to noun phrases (NP) rather than verb phrases (VP). Here we illustrate how a decision tree for $p(w_i \mid t)$ captures this. Some of the top nodes in this decision tree are shown in Figure 3. Each line gives a node number, Q — the question asked at that node, examples of answers, and probabilities for "of" and "in". Questions are specified by the question type numbers given in

Figure 4 in the next section. Looking at node 0 we see that the first question type is 1 — parent of the proposed word. The children trees are 1 and 2. We see that prepositions (IN) have been put in node 1. Since this is a binary choice, about half the preterminals are covered in this node. To get a feel for who is sharing this node with prepositions each line gives two examples. For node 1 this includes a lot of very different types, including NN (common singular noun).

Node 1 again asks about the preterminal, leading to node 4. At this point NN has split off, but NNS (common plural noun) is still there. Node 4 again asks about the preterminal, leading to node 12. By this point IN is only grouped with things that are much closer, e.g. RB (adverb).

Also note that at each node we give the probability of both "of" and "to" given the questions and answers leading to that node. We can see that the probability of "of" goes up from 0.05 at node 1 to 0.27 at node 16. The probabilities for "to" go in lockstep. By node 16 we are concentrating on prepositions heading prepositional phrases, but nothing has been asked that would distinguish between these two prepositions. However, at node 16 we ask the question "who is the grandparent" leading to nodes 39 and 40. Node 39 is restricted to the answer "noun phrase" and things that look very much like noun phrases — e.g., NX, a catchall for abnormal noun phrases, while 40 is restricted to PP's attaching to VP's and S's. At this point note how the probability of "of" dramatically increases for node 39, and decreases for 40.

That the tree is binary forces the decision tree to use information about words and nonterminals one bit at a time. In particular, we can now ask for a little information about many different previous words in the sentence.

We go from a single decision tree to a random forest by creating many trees, randomly changing the questions used at every node. First note that in our greedy selection of s_i's the outcome depends on the initial random assignment of a's. Secondly, each qt produces its own binary version q. Rather than picking the one that raises the data probability the most, we choose it with

1-6 The non-terminal of the parent, grandparent, parent[3], up to parent[6]

7-10 The previous non-terminal, 2-previous, up to 4-previous

11-14 The non-terminal just prior to the parent, 2-prior, up to 4 prior

15-16 The non-terminal and terminals of the head of the previous constituent

17-18 Same, but 2 previous

19-20 Same but previous to the parent

21-22 Same but 2 previous to the parent

23-24 The non-terminal and terminal symbols just prior to the start of the current parent constituent

25 The non-terminal prior to the grandparent

26 Depth in tree, binned logarithmically

27 Is constituent prior to parent a conjoined phrase.

Figure 4: Question types

probability m. With probability $1 - m$ we repeat this procedure on the list of q's minus the best. Given a forest of f trees we compute the final probability by taking the average over all the trees:

$$p(x \mid t) = \frac{1}{f} \sum_{j=1,f} p^j(x \mid t)$$

where p^j denotes the probability computed by tree j.

4 Implementation Details

We have twenty seven basic query types as shown in Figure 4. Each entry first gives identification numbers for the query types followed by a description of types. The description is from the point of view of the tree entry x to be added to the tree. So the first line of Figure 4 specifies six query types, the most local of which is

the label of the parent of x. For example, if we have the local context "(S (NP (DT the)" and we are assigning a probability to the preterminal after DT, (e.g., NN) then the parent of x is NP. Similarly one of the query types from line two is one-previous, which is DT. Two previous is ϵ, signifying nothing in this position.

Random forests, at least in obvious implementations, are somewhat time intensive. Thus we have restricted their use to the distribution $p(x \mid t)$. The forest size we picked is 4400 nodes. For the look-ahead probability, LAP, we use a single decision tree with greedy optimal questions and 1600 nodes.

We smooth our random forest probabilities by successively backing off to distributions three earlier in the decision tree. We use linear interpolation so

$$p_l(x \mid t) = \lambda(c_l) * \hat{p}_l(x \mid t) + (1 - \lambda(c_l)) * p_{l-3}(x \mid t)$$

Here p_l is the smoothed distribution for level l of the tree and $\hat{p}_l$ is the maximum likelihood (unsmoothed) distribution. We use Chen smoothing so the linear interpolation parameters λ are functions of the Chen number of the level l node. See (Chen and Goodman, 1996). We could back off to $l - 1$, but this would slow the algorithm, and seemed unnecessary.

Following (Klein and Manning, 2003) we handle unknown and rare words by replacing them with one of about twenty unknown word types. For example, "barricading" would be replaced by UNK-ING, denoting an unknown word ending in "ing." Any word that occurs less than twenty times in the training corpus is considered rare. The only information that is retained about it is the parts of speech with which it has appeared. Future uses are restricted to these pre-terminals.

Because random forests have so much latitude in picking combinations of words for specific situations we have the impression that it can overfit the training data, although we have not done an explicit study to confirm this. As a mild corrective we only allow verbs appearing 75 times or more, and all other words appearing 250 times or more, to be conditioned upon in question types 16, 18, 20, 22, and 27. Because the inner loop of random-forest training involves moving a conditioning event to the other decedent node to see if this raises training data probability, this also substantially speeds up training time.

Lastly Roark obtained the results we quote here with selective use of left-corner transforms (Demers, 1977; Johnson and Roark, 2000). We also use this technique but the details differ. Roark uses left-corner transforms only for immediately recursive NP's, the most common situation by far. As it was less trouble to do so, we use them for any immediately recursive constituent. However, we are also aware that in some respects left-corner transforms work against the fully-connected tree rule as operationalizing the "understand as you go along" cognitive constraint. For example, the normal sentence initial NP serves as the subject of the sentence. However in Penn-treebank grammar style an initial NP could also be a possessive NP as in (S (NP (NP (DT The) (NN dog) (POS 's)))) Clearly this NP is *not* the subject. Thus using left corner transforms on all NP's allows the parser to conflate differing semantic situations into a single tree. To avoid this we have added the additional restriction that we only allow left-corner treatment when the head words (and thus presumably the meaning) are the same. (Generally head-word rules dictate that the POS is the head of the possessive NP.)

5 Results and Analysis

We trained the parser on the standard sections 2-21 of the Penn Tree-bank, and tested on all sentences of length $\leq$ 100 of section 23. We used section 24 for development.

Figure 5 shows the performance of our model (last line, in bold) along with the performance of other parsers. The first group of results show the performance of standard parsers now in use. While our performance of 89.4% f-measure needs improvement before it would be worth-while using this parser for routine work, it has moved past the accuracy of the Collins-Bikel (Collins, 2003; Bikel, 2004) parser and is not statistically distinguishable from (Charniak, 2000).

	Precision	Recall	F
Collins 2003	88.3	88.1	88.2
Charniak 2000	89.6	89.5	89.6
C/J 2005	91.2	90.9	91.1
Petrov et.al. 2006	90.3	90.0	90.2
Roark 2001	87.1	86.6	86.8
C/R Perceptron	87.0	86.3	86.6
C/R Combined	89.1	88.4	88.8
This paper	**89.8**	**89.0**	**89.4**

Figure 5: Precision/Recall measurements, Penn Treebank Section 23, Sentence length ≤ 100

Conditioning Non-terminals	Conditioning Terminals	F-measure
8	1	86.6
10	2	88.0
13	3	88.3
17	4	88.8
21	6	89.0

Figure 6: Labeled precision-recall results on section 24 of the Penn Tree-bank. All but one sentence of length ≤ 100. (Last one not parsed).

The middle group of results in Figure 5 show a very significant improvement over the original Roark parser, (89.4% vs.86.8%). Although we have not discussed it to this point, (Collins and Roark, 2004) present a perceptron algorithm for use with the Roark architecture. As seen above (C/R Perceptron), this does not give any improvement over the original Roark model. As is invariably the case, when combined the two models perform much better than either by itself (C/R Combined — 88.8%). However we still achieve a 0.6% improvement over that result. Naturally, a new combination using our parser would almost surely register another significant gain.

In Figure 6 we show results illustrating how parser performance improves as the probability distributions are conditioned on more diverse information from the partial trees. The first line has results when we condition on only the "closest" eight non-terminal and the previous word. We successively add more distant conditioning

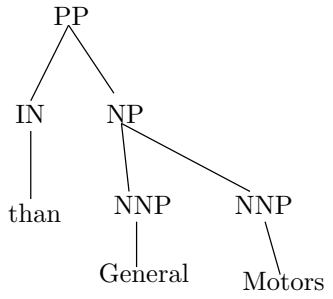

Figure 7: The start of an incorrect analysis for "than General Motors is worth"

events. The last line (89.0% F-measure) corresponds to our complete model but since we are experimenting here on the development set the result is not the same as in Figure 5. (The result is consistent with the parsing community's observation that the test set is slightly easier than the development set — e.g., average sentence length is less.)

One other illustrative result: if we keep all system settings constant and replace the random forest mechanism by a single greedy optimal decision tree for probability computation, performance is reduced to 86.3% f-measure. While this almost certainly overstates the performance improvement due to many random trees (the system parameters could be better adjusted to the one-tree case), it strongly suggests that nothing like our performance could have been obtained without the forests in random forests.

6 Conclusions and Future Work

We have presented a new top-down left-to-right parsing model. Its performance of 89.4% is a 20% error reduction over the previous single-parser performance, and indeed is a small improvement (0.6%) over the best combination-parser result. The code is publically available.[1] Furthermore, as models of this sort have received comparatively little attention, it seems reasonable to think that significant further improvements may be found.

One particular topic in need of more study is search errors. Consider the following example:

[1] http://bllip.cs.brown.edu/resources.shtml#software

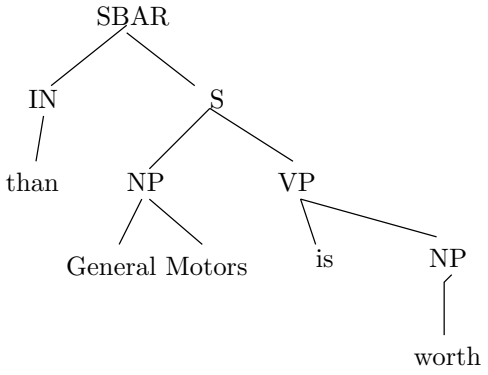

Figure 8: The correct analysis for "than General Motors is worth"

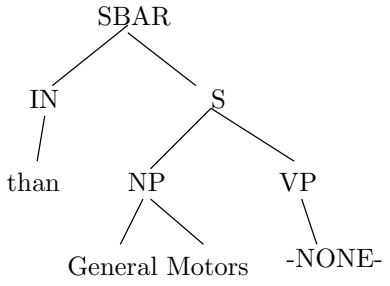

Figure 9: Alternative analysis for "than General Motors"

> The government had to spend more than General Motors is worth.

which is difficult for our parser. The problem is integrating the words starting with "than General Motors." Initially the parser believes that this is a prepositional phrase as shown in Figure 7. However, the correct tree-bank parse incorporates a subordinate sentential clause"than General Motors is worth", as in Figure 8. Unfortunately, before it gets to "is" which disambiguates the two alternatives, the subordinate clause version has fallen out of the parser's beam (unless, of course, one sets the beam-width to an unacceptably high level). Furthermore, it does not seem that there is any information available when one starts working on "than" to allow a person to immediately pick the correct continuation. It is also the case that the parsing model gives the correct parse a higher probability if it is available, showing that this is a search error, not a model error.

If there is no information that would allow a person to make the correct decision in time, perhaps people do not need to make this decision. Rather the problem could be in the treebank representation itself. Suppose we reanalyzed "than General Motors" in this context as in Figure 9. Here we would not need to guess anything in advance of the (missing) VP. Furthermore, we can make this change without loosing the great benefit of the treebank for training and testing. The change is local and deterministic. We can tree-transform the training data and then untransform before scoring. It is our impression that a few examples like this would remove a large set of current search errors.

Three other kinds of information are often added as additional annotations to syntactic trees: Penn-Treebank form-function tags, trace elements, and semantic roles. Most research on such annotation takes the parsing process as fixed and is solely concerned with improving the retrieval of the annotation in question. When they have been integrated with parsing, finding the parse and the further annotation jointly has not improved the parse. While it is certainly possible that this would prove to be the same for this new model, the use of random forests to integrate more diverse information sources might help us to reverse this state of affairs.

Finally there is no reason why we even need to stop our collection of features at sentence boundaries — information from previous sentences is there for our perusal. There are many known intra-sentence correlations, for example "sentences" that are actually fragments are much more common if the previous sentence is a question. The tense of sequential sentences main verbs are correlated. Main clause subjects are more likely to be co-referent. Certainly the "understanding" humans pick up helps them assign structure to subsequent phrases. How much, if any, of this meaning we can glean given our current (lack-of) understanding of semantics and pragmatics is an interesting question.

References

Y. Amit and D. Geman. 1997. Shape quantization and recognition with randomized trees. *Neural Computation*, 9:1545–1588.

Dan Bikel. 2004. *On the Parameter Space of Lexicalized Statistical Parsing Models*. Ph.D. thesis, University of Pennsylvania.

Leo Breiman. 2004. Random forests. *Machine Learning*, 45(1):5–32.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-fine n-best parsing and MaxEnt discriminative reranking. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics*, pages 173–180, Ann Arbor, Michigan, June. Association for Computational Linguistics.

Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *The Proceedings of the North American Chapter of the Association for Computational Linguistics*, pages 132–139.

Stanley F. Chen and Joshua Goodman. 1996. An empirical study of smoothing techniques for language modeling. In Arivind Joshi and Martha Palmer, editors, *Proceedings of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics*, pages 310–318, San Francisco. Morgan Kaufmann Publishers.

Michael Collins and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 111–118, Barcelona, Spain, July.

Michael Collins. 2003. Head-driven statistical models for natural language parsing. *Computational Linguistics*, 29(4):589–638.

A. Demers. 1977. Generalized left-corner parsing. In *Conference Record of the Fourth ACM Symposium on Principles of Programming Languages, 1977 ACM SIGACT/SIGPLAN*, pages 170–182.

James Henderson. 2003. Inducing history representations for broad coverage statistical parsing. In *Proceedings of HLT-NAACL 2003*.

Mark Johnson and Brian Roark. 2000. Compact non-left-recursive grammars using the selective left-corner transform and factoring. In *Proceedings of COLING-2000*.

Dan Klein and Christopher Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*.

Michell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing*.

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. 2006. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 433–440, Sydney, Australia, July. Association for Computational Linguistics.

Brian Roark. 2001. Probabilistic top-down parsing and language modeling. *Computational Linguistics*, 27(2):249–276.

Roger P.G. van Gompel and Martin J. Pickering. 2007. Syntactic parsing. In G. Gatskil, editor, *The Oxford handbook of psycholinguistics*. Oxford University Press.

Peng Xu and Frederick Jelinek. 2004. Random forests in language modeling. In *Proceedings of the 2004 Empirical Methods in Natural Language Processing Conference*. The Association for Computational Linguistics.