Node-Based Induction of Tree-Substitution Grammars

Rose Sloan
Yale University
rose.sloan@yale.edu

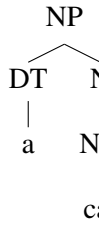
Abstract

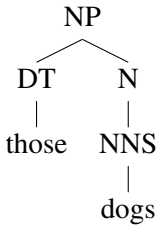
Because PCFGs are, as their name suggests, context-free, they cannot encode many dependencies that occur in natural language, such as the dependencies between determiners and nouns, allowing them to overgenerate phrases like *those cat*. One formalism that is able to capture many dependencies that PCFGs cannot is that of probabilistic tree-substitution grammars (PTSGs). Because PTSGs allow larger subtrees to be used as grammar rules, they can better model natural language but are also more difficult to induce from a corpus. In this paper, I will show how PTSGs can be used to represent dependencies between determiners and nouns and present a novel method for inducing a PTSG from a parsed corpus.

1 Introduction

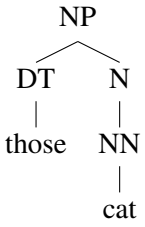
PCFGs are ill equipped to handle the case of simple noun phrases consisting of a determiner followed by a noun, as the grammaticality of these phrases is dependent on what types of nouns the determiner in question can precede. For example, while *the* can precede any noun, determiners like *a* and *another* can only occur with singular count nouns, while *those* can only precede plural nouns, and determiners like *more* can precede either plural nouns or mass nouns but not singular count nouns. Thus, the noun phrases *a book*, *more coffee*, and *those cards* are grammatical, whereas *a cards*, *more book*, and *those coffee* are not.

Representing these sorts of noun phrases with a PCFG is difficult. Consider the following toy corpus:

(1) a. 

b. 

A CFG representing this corpus would need to contain the rules NP $\rightarrow$ DT N, DT $\rightarrow$ those, N $\rightarrow$ NN, and NN $\rightarrow$ cat. While these are all reasonable rules, combining them gives us the following tree:

(2) 

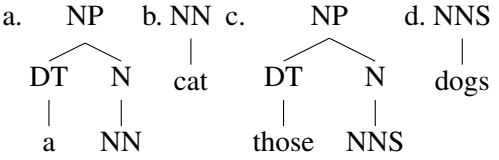
Thus, such a grammar would generate the blatantly ungrammatical noun phrase *those cat*. Furthermore, even to assign it a low probability, we would need a low probability for at least one of the rules that generated it, which would reduce the probability of at least one of the trees in (1). While we could perhaps mitigate this problem by removing the nodes labeled N and splitting the DT label into a set of labels corresponding to specific types of DTs, such a solution would overlook some generalizations (including, say, that *the* could precede any noun). Instead, we look for a formalism that represents these dependencies more naturally.

One such formalism is tree-substitution grammars (TSGs). Whereas CFG rules can be seen as one-level subtrees of the parse trees they generate, TSG rules can be any subtrees of these parse trees. A leaf node of a TSG rule whose label is not a lexical item is known as a substitution node, and parse trees can be built by replacing a substitution

Proceedings of the 12th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+12), pages 77–84,
Düsseldorf, Germany, June 29 - July 1, 2016.

node with a rule whose root has the same label as the substitution node.

Because TSGs allow for larger rules than CFGs, they can handle dependencies that CFGs do not. Consider, once again, the toy corpus presented in (1). While it is impossible to represent it using a CFG that does not overgenerate ungrammatical noun phrases, we could represent it with a TSG containing the following rules:

(3) a. 

b.

c.

d.

(3a) and (3b) can be combined to get (1a), and (3c) and (3d) can be combined to get (1b). It is no longer possible to generate (2), as the rule that produces *those* has a substitution node labeled NNS and thus can only accept plural nouns. (Similarly, the rule that produces *a* requires a singular count noun). Thus, TSGs allow us to accurately capture the dependencies between noun types and determiners.

While PTSGs present a more accurate model of natural language than PCFGs, they are also harder to induce from a corpus. Given a parse tree for a sentence, one can determine what CFG rules must have generated it simply by looking at each nonterminal node and its children. Then, given an entire treebank of parse trees, one can simply extract all the necessary CFG rules to produce that treebank and then get a PCFG by estimating the probability of each rule using one of a variety of techniques, the simplest of which involve simply counting the number of times each rule is used in producing the context. However, given a parse tree generated from a PTSG, it is less clear which rules formed it, as it is unknown which of the tree's internal nodes were substitution nodes in its derivation and which ones were already internal nodes in elementary trees. Furthermore, while some of the subtrees of the completed parse trees, such as the rules presented in (3), contain linguistically relevant information, others, such as many CFG rules or rules that simply memorize each full tree in the corpus, are not specific enough or overly specific and, as such, should have low probability or not be present in a PTSG at all.

In this paper, I will present a novel approach for inducing a PTSG from a parsed corpus, focusing specifically on PTSGs that model noun phrases such as the one in (3). This approach focuses on determining which nodes in the training set are substitution nodes. It does so by repeatedly sampling grammars from the training set. It then parses the data set with these grammars and uses the results of the parses to update the probability of trees' internal nodes being substitution nodes. This approach is simpler to understand and implement than other node-based approaches and does not require complex prior distributions.

2 Previous Approaches

Many previous approaches to TSG induction are data-oriented parsing (DOP) approaches that attempt to create grammars that include every possible rule that could have generated a corpus (Bod and Scha, 1996). In the most straightforward case, this means that the rules that comprise the tree-substitution grammar are simply all subtrees of all the trees in the training set. Unsurprisingly, these approaches lead to very large grammars, and in many cases, it is even necessary to transform each tree into some implicit representation (such as representing larger rules in terms of smaller rules instead of fully storing the trees) in order to store the grammar or at least to use it for parsing. Other approaches to data-oriented parsing try to limit the size of the grammar to some extent. For example, one approach, known as double-DOP, creates a grammar by taking every pair of parse trees from the training set and adding the largest subtree included in both trees (Sangati and Zuidema, 2011). This grammar is then interpolated with a CFG to create a grammar that can fully represent the training set and that includes all larger "productive" rules.

A number of more recent approaches, including the one presented in this paper, attempt not to find all TSG rules that could represent a corpus but to represent the corpus using the optimal distribution of TSG rules. One such approach called fragment grammars looks at TSGs from a generative perspective as a Bayesian model of the relative probability of productivity (forming novel phrases) and reuse (reusing previously constructed fragments) (O'Donnell et al., 2009). The mathematical model used to generate the grammar is a generalization of a Pitman-Yor Adaptor Grammar, a PCFG-based model that weakens the independence assumptions by introducing a vector of adaptor functions that map one probability distri-

bution over trees to another (Johnson et al., 2006). In Pitman-Yor Adaptor Grammars specifically (as opposed to adaptor grammars in general), these adaptor functions are based loosely on a Chinese restaurant process so that the distribution generated by the grammar reflects the outputs of a "rich get richer"-based process. Fragment grammars use these mathematical underpinnings within the PTSG framework, incorporating a "grow-child-or-not" term into the generative model, allowing it to assign and optimize the weights of elementary trees.

Similarly, approaches presented by Cohn et al. and by Post and Gildea use priors based on a Dirichlet process to obtain a grammar (Cohn et al., 2009; Post and Gildea, 2009). These approaches use Gibbs sampling to induce the grammar, and, like my approach, they focus on determining which nodes are substitution nodes, although the sampling methods I present are different.

3 Algorithm

3.1 Concept

Like the fragment grammars approach, my algorithm of node-based induction for inducing a PTSG attempts to find an optimal subset of subtrees. However, instead of explicitly representing probability distributions over grammars, I instead assign a probability to each internal node of each parse tree in the corpus, corresponding to the probability that the node is a substitution node. (For simplicity of notation, throughout this section, I will refer to this probability for node n as $p(n)$ or simply as node n's probability.) It is these probabilities that are optimized over the course of many iterations of training. Specifically, during each iteration, an intermediate PTSG is used to parse the training set, and the probability of each node is recalculated based on the probabilities of the different parses it generates.

Initially, the probability is set to the same value for every node in the training set. After trying values in the range [0.35, 0.9], I experimentally determined an initial probability of 0.55 for each node is most likely to result in a grammar that performs well on the test set. This is likely because this value imposes a slight prior against simply memorizing the training set. However, because the weighting of the different parses later in the algorithm tends to mitigate most of the bias introduced from the initial node probabilities, small changes in this initialization parameter have little effect on the final result.

3.2 Sampling and Parsing

To complete one iteration of training, we start by inducing a PTSG by randomly sampling from the parsed training set. Specifically, we decompose each tree, randomly choosing whether or not each internal node is a substitution node based on its probability at the start of the iteration (so that node n has probability $p(n)$ of being a substitution node in the decomposed tree). The set of trees resulting from this decomposition become our set of elementary trees, and we set the probability of each elementary tree using a relative frequency estimate (that is, simply letting the probability be the number of times that tree appears in the set of decomposed trees divided by the total number of trees with the same root node).

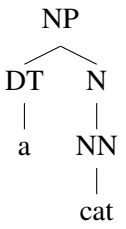
Once we have induced this PTSG, we then use it to parse each tree in the training set. In order to make parsing more efficient, any rules containing lexical items that do not appear in the phrase being parsed are removed before parsing with a standard CKY algorithm (Schabes et al., 1988). Doing so, we get all possible parses for the tree with our grammar, and we can then get an intermediate probability (p_{int}) for each node by examining the probabilities of the parses in which node n is a substitution node. This intermediate probability corresponds to the probability that the node is a substitution node when using this particular intermediate grammar. Furthermore, we assign a weight to each parse to prioritize parses in which some but not all of the nodes are substitution nodes (to discourage the model from doing something similar to simply inducing a PCFG or from memorizing entire trees). Specifically, in order to weight a parse more favorably the closer it is to having about half the internal nodes be substitution nodes, the weight of a parse is $\binom{t}{s}$ (t choose s) where s is the number of substitution nodes in a parse and t is the total number of internal nodes (i.e. the number of potential substitution nodes). Then, if $p(x)$ is the probability of a parse, $w(x)$ is the weight of a parse, S is the set of all parses in which n is a substitution node, and T is the set of all parses for the tree that n appears in, we can compute p_{int} using the following formula:

$$p_{int}(n) = \frac{\sum_{x \in S} w(x)p(x)}{\sum_{x \in T} w(x)p(x)}$$

3.3 An Example

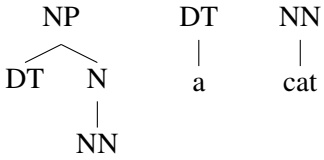
Consider the following tree from a hypothetical training set:

(4)

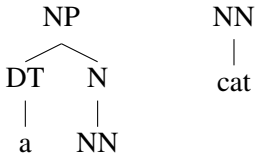


Let us assume that our intermediate grammar produced parses with the following three sets of elementary trees with probabilities p_1, p_2, and p_3 respectively:

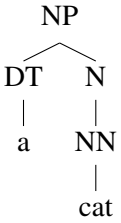
(5) a.



b.



c.



The parses in (2a) and (2b) both have weights of $\binom{3}{1} = \binom{3}{2} = 3$, as (5a) has 2 substitution nodes and (5b) has 1. However, the parse in (5c) only has weight $\binom{3}{0} = 1$, as it has no substitution nodes. Then, we can compute p_{int} for the node labeled DT, which is only a substitution node in (5a), as:

$$p_{int} = \frac{3p_1}{3p_1 + 3p_2 + p_3}$$

3.4 Updating Probabilities

Once we have calculated $p_{int}(n)$, we adjust the probability of node n by taking a weighted average of this intermediate probability and the probability from the start of the round, using the following formula:

$$p_{new}(n) = 0.6p_{old}(n) + 0.4p_{int}(n)$$

The higher p_{int} is weighted, the faster the node probabilities converge, but when p_{int} is weighted higher, each randomly selected grammar has a larger impact on the final grammar and thus could result in a final grammar that performs poorly on the test set. The precise weighting above was determined experimentally by running the algorithm a number of times with different weights to provide an optimal balance between allowing each round to significantly affect the node probabilities while still weighting p_{int} little enough so that a round in which the intermediate PTSG is chosen suboptimally will not derail the training process.

3.5 Getting the Final Grammar

We compute a convergence metric by examining how close our intermediate probabilities are to the node probabilities at the start of a round. We can say that node n has "converged" if the difference between $p_{old}(n)$ and $p_{int}(n)$ is less than 0.05. The convergence metric then is the number of "converged" nodes divided by the total number of internal nodes in all trees in the training set. If this number is over 0.95, training comes to an end and the node probabilities set at the end of the last round of training are used to sample the final grammar.

Once the node probabilities have been finalized, we decompose the training set 100 times using the same method we used in training. Then, the set of decomposed trees becomes the set of rules of our final PTSG, and, as before, probabilities are set using relative frequency estimates. Then, once we have determined the rules and probabilities for the final PTSG, we parse each rule in this PTSG using the other rules of the PTSG. If there is a parse for the rule made up of smaller rules and if the probability of this parse is greater than the probability of the larger rule, the rule is determined to be superfluous. Superfluous rules are removed from the grammar, and the probabilities are renormalized.

Finally, in order to account for unknown words, for each part of speech appearing in the training set, a tree with height 1 with a root labeled with the part of speech tag and with one leaf node labeled "unk" (short for "unknown") is added to the grammar. The probability of these rules is set according to the number of types and tokens for the part of

speech so that a part of speech with many distinct lexical items, such as count nouns, has a relatively high probability of unknown words compared to a part of speech with relatively few distinct lexical items, such as determiners. Specifically, the probability of the rule $POS \rightarrow unk$ was set to:

$$\frac{types(POS)}{types(POS) + tokens(POS)}$$

After adding these rules, the probabilities of all other rules whose roots are part of speech tags are renormalized.

4 Methods

4.1 Data Selection

The trees used for training and test sets are taken from the Adam portion of the Pearl-Sprouse corpus, a parsed version of the child-directed portions of the Brown subcorpus from CHILDES (Pearl and Sprouse, 2012; Brown, 1973; MacWhinney, 2000). Only noun phrases are examined, as they can be parsed quickly and are a structure for which a PTSG should be able provide an accurate model. Additionally, in order to allow the algorithm to distinguish between mass and count nouns, the NN label (the POS tag for singular nouns) corresponding to any mass noun is manually replaced by an NNM label. Similarly, to allow the algorithm to have rules applicable to all nouns, a node labeled simply N is inserted immediately above any node labeled NN, NNM, or NNS (the POS tag for plural nouns).

Furthermore, as the algorithm makes use of a CKY parser, any tree in the corpus which is not binary branching is modified to become right-branching. If an inserted node's children are both labeled N, it is labeled with N, so that the algorithm would treat compound nouns the same way as other nouns, and similarly, if the first child is labeled JJ (the POS tag for adjectives) and the second is labeled N, the inserted node is labeled N, as adjective-noun pairs distribute similarly to nouns in this dataset. All other inserted nodes are labeled by concatenating the labels of their children.

4000 noun phrases are then extracted from this modified corpus. None of these noun phrases include smaller internal noun phrases, so as to allow the algorithm to focus on dependencies between determiners and nouns, and all of them include at least one node labeled N (so as to eliminate single pronouns from the data set). They are also selected so that at least 30% of them contain mass nouns. 3200 of these nouns are randomly chosen to be the training set. The remaining 800 become the test set. Furthermore, every lexical item in the test set that does not appear in the training set was replaced with the word "unk" so that it can be properly parsed by the induced grammar.

4.2 Tests Run

The first baseline the induced grammar is tested against was a PCFG. The rules of the grammar are taken from all the PCFG productions in the training, and the probabilities are set using relative frequency estimates. Furthermore, rules going from each part of speech to "unk" are added with probabilities set the same way as they were in the PTSG so that trees with unseen lexical items can be parsed.

The second baseline is a PTSG whose rules are simply the full trees in the training set. The probability of each rule is set using a relative frequency estimate, so the probability of a given tree is simply the number of times the tree appears in the training set divided by the total number of trees in the training set.

The third baseline is a PTSG obtained by randomly sampling from the training set, specifically by decomposing each tree 100 times and setting the probabilities using relative frequency estimates, just as at the end of the induction algorithm. However, instead of using the trained probabilities, while sampling, the probability of each node being a substitution node is simply set to its initial probability of 0.55. To make this more comparable to the induced grammar, rules going from each POS tag to "unk" are added with their probabilities equal to their probabilities in the induced PTSG, and all other rules' probabilities are renormalized.

Lastly, Sangati and Zuidema's code for double-DOP is run on the training set to obtain their set of fragments and CFG rules with counts. Using these counts, probabilities for each rule are obtained using relative frequency estimates. Then the same rules for unknown lexical items with the same probabilities as in the induced grammar were added, and the probabilities are renormalized. Other previous approaches were not tested because of the difficulty of finding a working implementation of them.

Finally, to avoid zero probabilities, especially

Method	Training	Test	Grammar Size
Node-Based	-25263	-6770	1359
PCFG	-30905	-7091	990
Full Trees	-22280	-6814	1572
Sampling	-30135	-7266	1721
Double-DOP	-28882	-7032	2404

Table 1: Log probabilities of training and test sets on different grammars

for the full trees baseline, when computing the probability of a tree in the test set with the PTSGs obtained through node-based induction, sampling, and taking full trees, we also parse it with the PCFG induced for the first baseline. (This is not necessary for double-DOP, as the algorithm for double-DOP already incorporates all possible CFG rules.) The probability of the tree is then calculated to be a weighted average of the two probabilities, with the PCFG weighted at 0.05, while the PTSG is weighted at 0.95. Any trees in the test set that cannot be parsed with the PCFG are removed from the test set and ignored.

5 Results

Table 1 shows the results for how node-based induction compares to the baselines with a training set of size 3200 and a test set of size 793. (Initially, the test set was of size 800, but 9 noun phrases were removed because they contained structures unseen in the training set and thus could not be parsed by any of the grammars.) The numbers provided here are obtained by summing the log probabilities of the best parses for each tree in the data set. (In every case except for the PCFG baseline, these probabilities are also computed by taking a weighted average of the probability of the best parse with the chosen model and the best parse with a PCFG, as explained in the methods section). Thus, larger (i.e. less negative) numbers correspond to higher probabilities and therefore better results.

These results demonstrate that, apart from simply memorizing the training set (and grossly overfitting), the PTSG induced by node-based induction assigns the highest probability to the training set. Additionally, when tested on an unseen test set, node-based induction outperforms each of the baselines. It is also worth noting that when sampling randomly without first training the substitution node probabilities, the resulting grammar performs nearly as badly as a PCFG on the training set and worse than all other grammars on the test, thus demonstrating that the optimization of the substitution node probabilities is in fact what allows node-based induction to produce a well-performing grammar. It is also worth noting that node-based induction produces the smallest grammar except for the PCFG, making it faster to parse with.

In order to determine how well the induce PTSG models the distribution of nouns and determiners, all 1442 noun phrases of the format "determiner noun" were extracted from the training set and, for each determiner that appeared more than 5 times, the probability distribution over different types of nouns occurring with that determiner was computed. These distributions are shown in table 2. Then, 1442 noun phrases of the form "determiner noun" were generated using the PTSG induced with node-based induction, and the same distributions were computed, shown in table 3. The same was done for the PCFG. Then, the Kullback-Leibler divergence was computed between the distributions generated from each of the PTSG induced through node-based induction and the PCFG and the true distribution from the training corpus, using add-one smoothing to avoid zero probabilities. These values are shown in table 4.

Determiner	Count Noun	Mass Noun	Plural Noun
a	0.983	0.015	0.002
an	0.952	0.048	0.000
another	0.714	0.286	0.000
any	0.048	0.714	0.238
no	0.571	0.286	0.143
some	0.000	0.913	0.087
that	0.857	0.143	0.000
the	0.712	0.230	0.058
this	0.960	0.040	0.000

Table 2: Probability distributions of noun types cooccurring with common determiners in the training set

These results show that, while the distributions produced by the node-based PTSG are not as strongly skewed as the empirical distributions, where many of the probabilities are over 0.9, they do reflect dependencies between determiners and noun types. (This may also reflect that, even in the empirical distributions, none of the probabilities are 1, reflecting the presence of noun phrases like

Determiner	Count	Mass	Plural
a	0.82	0.15	0.03
an	0.71	0.18	0.11
any	0.32	0.53	0.16
some	0.18	0.79	0.03
that	0.85	0.10	0.05
the	0.74	0.21	0.05
this	0.86	0.03	0.10

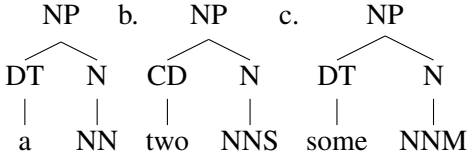
Table 3: Probability distributions of noun types cooccurring with common determiners in noun phrases generated by the PTSG

Determiner	Node-Based	PCFG
a	0.13	0.40
an	0.11	0.80
any	0.16	0.52
some	0.18	1.03
that	0.03	0.09
the	0.00	0.03
this	0.05	0.33

Table 4: K-L divergences of noun phrases generated by the node-based PTSG and the PCFG compared to the empirical distribution

another coffee where a noun that would normally be a mass noun serves as a coffee.) Furthermore, the K-L divergences are much smaller than those generated by the PCFG, a formalism that cannot encode these dependencies.

Furthermore, qualitatively speaking, many of the elementary trees that appear in the grammar induced by node-based induction make linguistic sense, such as those below:

(6) a. 

b.

c.

(6a) represents that *a* only appears before count nouns. (6b) represents that *two* appears before plural nouns. (6c) represents that *some* generally appears before mass nouns. Furthermore, to account for the fact that *some* can also appear before plural nouns (which are rarer in the data set than mass nouns) and even count nouns in limited grammatical contexts (as in sentences like *some person will like this*), there is another elementary tree in the grammar identical to (6c) but without the NNM node (so that N is a substitution node). However, this tree's probability is an order of magnitude lower than the tree in (6c), indicating that *some* appears primarily but not exclusively before mass nouns. Other rules indicate that the induced grammar learns several common compound nouns, including *cookie dough*, *rubber band*, and *trash can*, as single rules (instead of requiring each of the nouns to individually be substituted into a N $\rightarrow$ N N rule, as would be the case in a CFG).

6 Conclusion

In this paper, I have presented a novel approach for induction of probabilistic tree substitution grammars, which represents the probability distribution over possible tree-substitution grammars by assigning probabilities to potential substitution nodes and determines the optimal probabilities through repeated sampling and parsing. This approach is able to produce grammars that accurately represent dependencies between determiners and nouns, including, for example, elementary trees that require *a* to appear before a count noun. Furthermore, these grammars produce higher probability parses than standard PCFGs when tested on an unseen test set and also outperform a purely sampling-based approach where the probabilities assigned to the substitution nodes are not optimized.

Here, I have shown that tree-substitution grammars induced through node-based induction can more accurately represent the probabilities of potential parses for non-recursive noun phrases than traditional PCFG-based approaches or grammars induced from DOP-based approaches. We have not yet run experiments testing this algorithm on structures beyond noun phrases, but future work could adapt this algorithm to work with larger grammatical structures, including full sentences, and it could then be used to induce grammars that more accurately model language and generate more accurate parses.

References

Rens Bod and Remko Scha. 1996. Data-oriented language processing: An overview. *Computing Research Repository*.

Roger Brown. 1973. *A first language: The early stages.* Harvard U. Press.

Trevor Cohn, Sharon Goldwater, and Phil Blunsom. 2009. Inducing compact but accurate tree-substitution grammars. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 548–556. Association for Computational Linguistics.

Mark Johnson, Thomas L. Griffiths, and Sharon Goldwater. 2006. Adaptor grammars: A framework for specifying compositional nonparametric bayesian models. In *Advances in neural information processing systems*, pages 641–648.

Brian MacWhinney. 2000. *The CHILDES project: The database*, volume 2. Psychology Press.

Timothy J. O'Donnell, Noah D. Goodman, and Joshua B. Tenenbaum. 2009. Fragment grammars: Exploring computation and reuse in language. Technical Report MIT-CSAIL-TR-2009-013, Massachusetts Institute of Technology.

Lisa Pearl and Jon Sprouse. 2012. Computational models of acquisition for islands. *Experimental syntax and island effects*, pages 109–131.

Matt Post and Daniel Gildea. 2009. Bayesian learning of a tree substitution grammar. In *Proceedings of the ACL-IJCNLP 2009 Conference Short Papers*, pages 45–48.

Federico Sangati and Willem Zuidema. 2011. Accurate parsing with compact tree-substitution grammars: Double-DOP. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 84–95. Association for Computational Linguistics.

Yves Schabes, Anne Abeille, and Aravind K. Joshi. 1988. Parsing strategies with 'lexicalized' grammars: Application to tree adjoining grammars. In *Proceedings of the 12th Conference on Computational Linguistics - Volume 2*, pages 578–583.