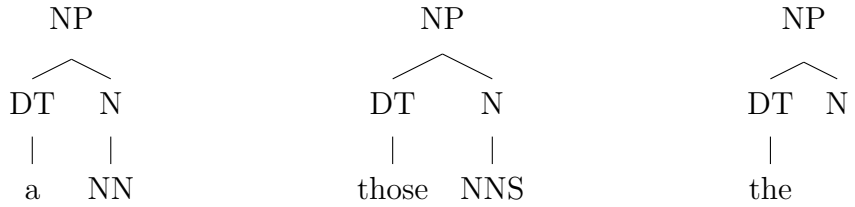
Node-Based Induction of Tree-Substitution Grammars

Rose Sloan
Advised by Robert Frank
April 20, 2016

Abstract

Traditionally, syntactic parsing is done using probabilistic context-free grammars (PCFGs) or variants thereof, as there are standard efficient methods for parsing with PCFGs and for extracting them from a corpus. However, PCFGs do not accurately represent many dependencies in natural language. For example, many determiners can only occur with certain types of nouns. 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. To represent these dependencies with a PCFG, we must have many separate categories for both determiners and nouns, as a simple rule like NP $\rightarrow$ DT N (where DT stands for determiner and N stands for noun) will overgenerate noun phrases like *a water* or *those cat*.

One formalism that makes representing long-range dependencies much simpler is probabilistic tree-substitution grammars (PTSGs). While every CFG rule can be seen as one level of a syntactic parse tree, and thus a subtree of height 2, TSG rules can be any subtree of a syntactic tree, thus allowing them to concisely represent dependencies that involve more than one level of syntactic structure. For example, a TSG for generating noun phrases can represent what types of nouns common determiners can precede using rules like the following:



In this thesis, I propose a method for inducing tree-substitution grammars from a parsed corpus, in which I determine what nodes are substitution nodes by parsing the training set using a series of randomly sampled PTSGs and looking at the substitution nodes in the most probable parses. I also compare my approach to the previous approach of data-oriented parsing, an approach that seeks to find all possible TSG rules that could represent a data set and thus results in very large grammars, as well as a number of simple baselines. Specifically, I examine how each of these approaches performs when trained on a set of noun phrases from child-directed speech.

Contents

Chapter 1

Introduction

1.1 Syntactic Parsing

Syntactic parsing is the problem of taking in a string of words and generating a parse tree that represents the string's syntactic structure. This is done using some formal set of rules that represent how a string could be generated, known as a grammar. One simple and very common grammar formalism is that of a probabilistic context-free grammar (PCFG). A context-free grammar is a 4-tuple (N, T, S, R) where N is a set of nonterminals, T is a set of terminals, S is an element of N that serves as a start symbol, and R is a set of rules, all of which are ordered pairs in the set $N \times (N \cup T)^*$ (Collins, 2003). (These rules are usually written in the format $A \rightarrow \beta$, where A is a nonterminal and β is a string of terminals and nonterminals.) The set of parse trees generated by a grammar are the trees whose roots are the start symbol, whose leaves are all terminals, and whose non-leaf nodes are nonterminals whose children correspond to the righthand side of a rule whose lefthand side is that nonterminal. In the domain of natural language, typically terminals correspond to words whereas nonterminals correspond to grammatical categories. Thus, a typical rule might be S $\rightarrow$ NP VP, demonstrating that sentences can branch into a noun phrase and verb phrase.

Probabilistic context-free grammars differ from context-free grammars only in that they additionally assign a probability to each rule so that the probabilities of all the rules whose lefthand side is a given terminal sum to 1. Thus, this probability represents the odds of expanding that nonterminal with that rule. Then, we can also assign a probability to each parse tree generated by a PCFG by multiplying together the probabilities of all the rules used to generate the tree. Thus, if a PCFG can generate multiple parse trees for the same sentence, we can decide which one is most likely by determining which one has the highest probability. This allows PCFGs to return a single best parse even when parsing ambiguous languages.

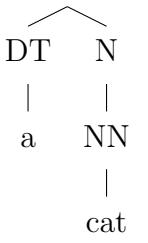
PCFGs are simple to induce and to use for parsing, and as such, they are commonly used in parsing. However, they suffer, because they are, as their name suggests, context-free. The rule used to expand any given nonterminal is independent of all other nodes in the tree. (In fact, it is independent of everything but the identity of the node itself.) However, this independence assumption does not hold in natural language, which contains a vast number of short and long range dependencies. As such, PCFGs, especially when using standard grammatical categories as nonterminals, fail to capture many common linguistic phenomena.

1.2 Nouns and Determiners

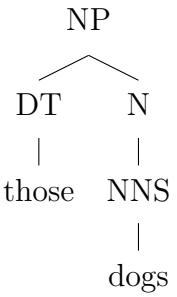
In English, one case that PCFGs are ill equipped to handle is the case of simple noun phrases consisting of a determiner followed by a noun, as the grammaticality of these phrases is dependent on what types of nouns the determiner in question can precede. For example, while *the* can precede any noun, 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.



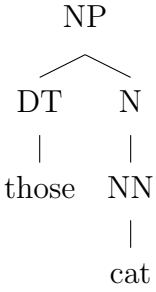
b.



(Here, DT stands for determiner, N stands for any noun, NN specifically refers to singular count nouns, and NNS refers to plural nouns.)

A CFG representing this corpus would need to contain the rules NP → DT N, DT → those, N → NN, NN → 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.

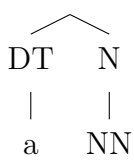
1.3 Tree-Substitution Grammars

One such formalism is the formalism of 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 node with a rule whose root has the same label as the substitution node.

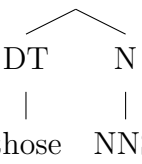
Formally speaking, a tree-substitution grammar is also a 4-tuple (N, T, S, R), where N is a set of nonterminals, T is a set of terminals, S is a nonterminal start symbol, and R is a set of rules (Cohn et al., 2009). However, instead of the context-free rules allowed by CFGs, in a TSG, the elements of R are all elementary trees whose internal nodes are all labeled with nonterminals and whose leaf nodes can either be terminals or nonterminals. The nonterminal leaf nodes are the substitution nodes. To generate a valid parse tree with a TSG, we begin with an elementary tree whose root node is S. Then, we can replace any substitution node with another elementary tree whose root has the same label as the substitution node. We repeat this process until all leaf nodes have terminal labels in order to get a complete parse tree. As with context-free grammars, we can also get a probabilistic tree-substitution grammar (PTSG) by assigning a probability to each elementary tree so that the probabilities of all the elementary trees whose root is a given nonterminal will always sum to 1.

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 the CFG rules discussed in section 1.2 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 thesis, I give a brief overview of methods used to induce PTSGs from parsed corpora and present a novel approach for doing so.

Chapter 2

Previous Approaches

2.1 Data-Oriented Parsing

Data-oriented parsing (DOP) approaches are approaches that attempt to create grammars that include every possible rule that could have generated a corpus (Bod & 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. The probabilities of these trees can then be set using a variety of approaches, ranging from relative frequency estimation (simply counting up the number of times a subtree appears in the corpus and then normalizing) to more complicated statistical approaches. 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. In the simplest case, this means that instead of simply taking every subtree that appears in the corpus, to generate the grammar, one randomly samples a certain number of subtrees from the corpus and uses these sampled trees as the rules of the grammar (again, setting the probabilities based on the number of times each tree was sampled using one of a variety of simple or complex statistical approaches). This approach will result in grammars that will exclude most large unproductive rules, but, because smaller trees are significantly more frequent than larger trees, this may miss large but linguistically salient trees. Other approaches take a more systematic approach to restricting the size of the grammar. 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 & 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.

It is worth noting that even the most restrictive DOP approaches are still, at their basic level, trying to produce a grammar that contains all rules that could have generated the corpus. While what "could have generated" means varies depending on the implementation, in every case, DOP approaches try include as many potential rules as possible in the final grammars instead of taking some optimized subset.

2.2 Adaptor Grammars

One formalism that weakens the independence assumptions of PCFGs in a way similar to PTSGs is the formalism of adaptor grammars (Johnson et al., 2006). Adaptor grammars rely on seeing PCFGs as distributions over trees. In a standard PCFG, the distribution of trees rooted in a given nonterminal is dependent only on the context-free rules and the distributions of potential children. Formally speaking, we can define G_X as the distribution over trees rooted at X and $TREEDIST_A(G_{B_1}, \ldots, G_{B_k})$ as the probability distribution over all trees rooted at A with k children rooted at B_1 through B_k, each generated independently from its own probability distribution G_{B_i}. Then, if $\theta_{X \to Y}$ is the probability of the rule $X \to Y$, we can write:

$$G_A = \sum_{A \to B_1 \ldots B_k \in R} \theta_{A \to B_1 \ldots B_k \in R} TREEDIST_A(G_{B_1}, \ldots, G_{B_k})$$

Adaptor grammars loosen this independence assumption by introducing an adaptor C, which is a vector of functions from distributions over trees to other distributions, with one function corresponding to each nonterminal. Then, C_A is the adaptor function corresponding to the nonterminal A, which maps distributions over trees rooted at A to other distributions of trees rooted at A. We can now define a second distribution $H_A = C_A(G_A)$ and modify our definition of G_A so that:

$$G_A = \sum_{A \to B_1 \ldots B_k \in R} \theta_{A \to B_1 \ldots B_k \in R} TREEDIST_A(C_{B_1}(G_{B_1}), \ldots, C_{B_k}(G_{B_k}))$$

If the adaptor functions are the identity function, this is just a PCFG. With other adaptor functions, this can represent different types of probability distributions.

One type of adaptor grammar, known as a Pitman-Yor adaptor grammar operates under a

"rich get richer" concept. While the actual model is mathematically highly complex, the basic idea behind it is that of a Chinese restaurant process, a stochastic process in which some option (a "table" in the Chinese restaurant metaphor) is chosen at each time step, and at time n, the probability of choosing a new option (an "empty table") is $\frac{1}{n+1}$ while the probability of choosing any previously chosen option is proportional to the number of times it has been chosen already (the number of "people at the table").

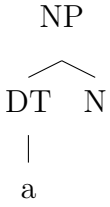
In the context of generative grammar, this means that initially, the distribution is determined by a PCFG. However, as sentences are generated, the subtrees generated from each nonterminal are cached. (These are not TSG-style elementary trees but instead the entire subtree rooted at a given nonterminal.) Then, when generating a tree, when expanding a nonterminal, with some probability, it is expanded with a PCFG rule to form a potentially new phrase, and otherwise, it is expanded using one of the cached subtrees.

2.3 Fragment Grammars

Fragment grammars take much of the structure of Pitman-Yor adaptor grammars and incorporate it in a framework closer to tree-substitution grammars. Unlike DOP, the goal of fragment grammars is not to find all TSG rules that could represent a corpus but to optimally represent the corpus using TSG-style rules (O'Donnell et al., 2009). This is done by looking at trees from a generative perspective as a Bayesian model of the relative probability of productivity (forming novel phrases) and reuse (reusing previously constructed fragments). In particular, as with Pitman-Yor adaptor grammars, productivity and reuse of grammatical fragments are represented as a form of stochastic memoization, a technique used commonly in programming (specifically dynamic algorithms). Memoization is a technique by which the results of smaller pieces of computation (formation of syntactic trees in this context) are memorized and reused.

As with a Pitman-Yor adaptor grammar, in a fragment grammar, the distribution of the ways to expand any given node adapts based on a Chinese restaurant process where previously seen trees are cached and reused. However, whereas the adaptor functions in an adaptor grammar are conditioned only on the identity of the node being expanded, fragment grammars instead look at so-called tree prefixes, which are equivalent to the elementary trees used as TSG rules. For example, consider the following elementary tree:

(1)



In a Pitman-Yor adaptor grammar, this tree would be completed either using a CFG rule whose lefthand side was N or by using one of the cached subtrees whose root is N (with the relative probabilities determined by the Chinese restaurant process). A fragment grammar is similar. However, if one of the cached subtrees is used, then instead of just using a subtree whose root is N, the model would use a cached subtree that had previously been substituted into the tree in (1). The optimal set of tree prefixes to use are then determined through repeated sampling, using a "grow-child-or-not" procedure to determine whether or not to expand a tree prefix.

Chapter 3

Node-Based Induction

3.1 The Algorithm

3.1.1 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 that node is a substitution node. (For simplicity of notation, throughout this section, I will be referring 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.

3.1.2 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 decomposed trees becomes our set of elementary trees for the PTSG, 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).

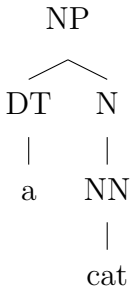
Once we have induced this PTSG, we then use it to parse each tree in the training set. In order to make parsing more efficient, any rules containing lexical items that do not appear in the phrase being parsed are removed before parsing with a standard CKY algorithm (Schabes et al., 1988). We can then 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 $\binom{t}{s}$ 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, mathematically speaking, 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:

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

3.1.3 An Example

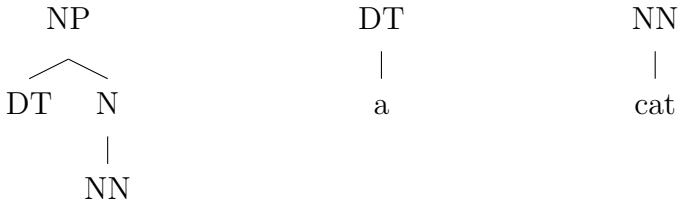
Consider the following tree from a hypothetical training set:

(1)



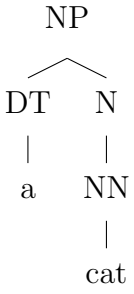
Let us assume that our intermediate grammar produced parses with the following three sets of elementary trees with probabilities p_1, p_2, and p_3 respectively:

(2) a.



b.

c.



The parses in (2a) and (2b) both have weights of $\binom{3}{1} = \binom{3}{2} = 3$, as (2a) has 2 substitution nodes and (2b) has 1. However, the parse in (2c) 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 (2a), as:

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

3.1.4 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 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.

3.1.5 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"). The probability of these rules was 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, would have 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 were part of speech tags were renormalized.

3.2 Data Selection

The trees used for training and test sets were 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 & Sprouse, 2012). 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 was manually replaced by an NNM label. Similarly, to allow the algorithm to have rules applicable to all nouns, a node labeled simply N was 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 was not binary branching was modified to become right-branching. If an inserted node's children were both labeled N, it was labeled with N, so that the algorithm would treat compound nouns the same way as other nouns, and similarly, if the first child was labeled JJ (the POS tag for adjectives) and the second was labeled N, the inserted node was labeled N, as adjective-noun pairs distribute similarly to nouns in this dataset. All other inserted nodes were labeled by concatenating the labels of their children.

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

3.3 Tests Run

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

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

The third baseline was 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 was 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" were added with their probabilities equal to their probabilities in the induced PTSG, and all other rules' probabilities were renormalized.

Lastly, Sangati and Zuidema's code for double-DOP was run on training set to obtain their set of fragments and CFG rules with counts. Using these counts, probabilities for each rule were 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 were renormalized.

Finally, to avoid zero probabilities, especially 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 was not necessary for double-DOP, as the algorithm for double-DOP already incorporated 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.

Chapter 4

Results

Table 4.1 displays 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 7 noun phrases were removed because they contained structures that were unseen in the training set and thus could not be parsed by any of the grammars.) The numbers provided here were 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 were 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.

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 4.1: Log probabilities of training and test sets on different grammars

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 set, thus demonstrating that the optimization of the substitution node probabilities is in fact what allows node-based

induction to produce a well-performing grammar. Furthermore, except for the PCFG, node-based induction produces the smallest grammar and therefore most efficient to use for parsing.

Additionally, in the grammar induced through node-based induction, about 21% of the probability mass for noun phrases was reserved for previously seen full phrases memorized by the system, and the rest was reserved for potentially novel noun phrases composed of smaller elementary trees. This indicates that, while the grammar does memorize commonly occurring noun phrases, it has strong generative capacities.

In order to gauge whether the phrases the induced grammar generated were grammatical, all 1442 noun phrases of the format "determiner noun" were extracted fro 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 4.3. 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 4.4. 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 4.2.

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

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 *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.

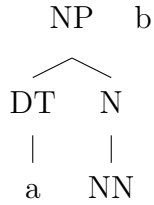
Furthermore, qualitatively speaking, many of the elementary trees that appear in the grammar induced by node-based parsing make linguistic sense, such as those in (1).

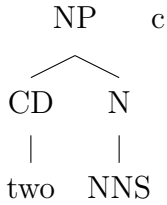
Determiner	Count	Mass	Plural
a	0.983	0.015	0.002
an	0.952	0.048	0.000
another	0.714	0.286	0.000
any	0.048	0.714	0.238
no	0.571	0.286	0.143
some	0.000	0.913	0.087
that	0.857	0.143	0.000
the	0.712	0.230	0.058
this	0.960	0.040	0.000

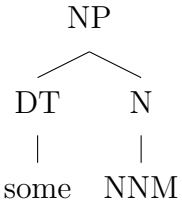
Table 4.3: Probability distributions of noun types cooccurring with common determiners in the training set

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

(1) a. 

b. 

c. 

(1a) represents that *a* only appears before count nouns. (1b) represents that *two* appears before plural nouns. (1c) 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 (1c) 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 (1c), 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).

Chapter 5

Conclusion

In this thesis, 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 the grammars induced using the double-DOP approach.

Syntactic parsing is an important problem in natural language processing, which has applications to a number of practical problems ranging from machine translation to question answering. 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 DOP-based approacheds. If future work were to adapt this algorithm to work with larger grammatical structures, including full sentences, it could be used to induce grammars that more accurately model language and generate more accurate parses.

Bibliography

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

Cohn, Trevor, Sharon Goldwater & 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*, 548–556. Association for Computational Linguistics.

Collins, Michael. 2003. Head-driven statistical models for natural language parsing. *Computational linguistics* 589–637.

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

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

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

Sangati, Federico & Willem Zuidema. 2011. Accurate parsing with compact tree-substitution grammars: Double-DOP. In *Proceedings of the conference on empirical methods in natural language processing*, 84–95. Association for Computational Linguistics.

Schabes, Yves, Anne Abeille & 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*, 578–583.