Grammar Induction with Prototypes Derived from Interlinear Text

Ryan Georgi

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Arts

University of Washington

2009

Program Authorized to Offer Degree: Computational Linguistics

University of Washington
Graduate School

This is to certify that I have examined this copy of a master's thesis by

Ryan Georgi

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

_____
Fei Xia

_____
William Lewis

Date: _____

Signature_____

Date_____

University of Washington

Abstract

Grammar Induction with Prototypes Derived from Interlinear Text

Ryan Georgi

Chair of the Supervisory Committee:
Prof Fei Xia
Computational Linguistics

In this thesis, we propose that instances of interlinear glossed text (IGT), as found in a wide range of linguistic papers, represent enriched content similar to partially annotated corpora. With such a type of data readily available for many languages for which little to no other data is available, we attempt to create a system which utilizes this information to bootstrap grammar induction techniques.

The grammar induction system we have developed consists of an implementation of the EM algorithm modified following (Haghighi & Klein 2006) to use intuitive rules known as prototypes to constrain the labels chosen by the algorithm. We investigate the possibility of using syntactic projection on IGT instances following (Xia & Lewis 2007) to automatically derive prototypes, which may then be fed into the modified inside-outside algorithm to inform the constituent grouping and labeling made by the algorithm.

The system developed in this thesis shows results on German IGT instances extracted from the ODIN Database (Lewis 2006) that suggest though further refinement is needed, this method of prototype extraction may be viable for developing a constrained grammar-induction algorithm for multiple low-density languages.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

CCM: **C**onstituent **C**ontext **M**odel: an unsupervised bracket-inducing algorithm (Klein & Manning 2002).

CFG: **C**ontext-**F**ree-**G**rammar. A formal grammar that consists of nonterminal symbols Σ, terminal symbols θ, and rewrite rules R.

EM: **E**xpectation **M**aximization. A class of algorithms designed to maximize the likelihood of a model.

IGT: **I**nterlinear **G**lossed **T**ext: a form of gloss notation used for foreign languages in linguistic papers, often using three lines for source language, semantic gloss, and English translation.

INSIDE-OUTSIDE ALGORITHM: Implementation of an EM algorithm as it applies to parse trees.

LANGUAGE DENSITY: Availability of (primarily) electronic resources for a language. English, Chinese, and French are examples of High-Density languages, while Hausa, Warlpiri, and Tagalog are examples of low-density languages (Gordon 2005), (Maxwell & Hughes 2006).

NONTERMINAL: In a parse tree, any node that is not a leaf node.

PCFG: **P**robabalistic **C**ontext-**F**ree **G**rammar. A Context-Free Grammar that also incorporates rule probabilities. Sometimes referred to as SCFG, for **S**tochastic CFG.

POS: **P**art **o**f **S**peech. As opposed to **syntactic** or phrasal tags, POS tags mark individual tokens.

PRETERMINAL: In a parse tree, the nodes immediately dominating the terminals.

TERMINAL: In a parse tree, the "leaf" nodes at the bottom. May be either a POS tag or word.

YIELD: The complete ordered series of terminal nodes that are all descendents of a given nonterminal.

Chapter 1

INTRODUCTION

Syntactic trees can be a valuable source of information about a language, capturing information phrase order, constituency, and well-formedness rules. Syntactic parsing is an important path of study in the field of computational linguistics. A properly parsed set of linguistic data can serve as a bootstrap for further linguistic work such as attaching semantic meaning, identifying named entities, and other linguistic tasks with requirements for deep analysis.

Since the introduction of electronic treebanks in the last decade, there has been substantial work in producing supervised parsers and other tools extracted from these enriched data sources. While invaluable tools in producing high-performance parsers and other tools, the creation of these treebanks is incredibly expensive, and thus they only exist for a handful of the world's languages (Marcus, Santorini & Marcinkiewicz 1994) (Maamouri, Bies, Buckwalter & Mekki 2004) (Ircs 2002). This small group of languages, such as English, French and Chinese, dominate the field of computational linguistics.

At the time of this writing, most state-of-the-art parsers rely on such supervised resources and are thus limited to languages with sufficient resources. Many languages, though they may have large speaker populations, have limited annotated resources available. Russian, despite having over a hundred million native speakers (Gordon 2005) and being an extremely prolific language in the world of literature, falls into this category. Others, such as Tamil, have small speaker populations, with little to no available resources. The goal of this thesis is to find a general method for producing parsers for languages with few resources with minimal human input.

1.1 Grammar Induction

In order to address both the cost of systems built upon expensive treebanks and scarcity of resources for lower-density languages, there has been a good deal of work in recent years on developing unsupervised parsers using machine learning techniques which can programmatically induce grammars from unannotated corpora. Within the literature, work usually falls into one of two approaches:

- Iteratively refining algorithms, such as the inside-outside algorithm, a flavor of EM designed for PCFGs.
- Distributional methods, such as k-ways clustering to classify commonly co-occurring strings in categories.

Individually, each of these methods has drawbacks. Inside-Outside re-estimation, as with many Expectation Maximization problems with large parameter spaces, contains many local optima and it is hard to ensure that the model found via this algorithm will be correct. Furthermore, the goal of EM algorithms is not to find the best linguistically motivated model to explain the data, but rather the model that maximizes the likelihood of the data. This goal may have the unfortunate consequence of producing a grammar that is predictive of the data, but that may be linguistically vacuous. For instance, EM re-estimation may result in a grammar where the symbols NP and VP are swapped, resulting in CFG rules such as VP $\rightarrow$ DT NN and NP $\rightarrow$ VB PP.

Distributional methods, while they can be formulated to group commonly co-occurring words into constituents quite well, have even greater issues with symbol confusion. While distributional algorithms find patterns in the data on their own, mapping the similar structures to existing tag labels is not automatically possible.

Because inside-outside re-estimation and distributional approaches model slightly different aspects of natural language yet aim to achieve the same goal, combining the two may produce better results than running each individually.

1.2 Prototyping as Bootstrapping

Despite the best attempts of these algorithms, there are ultimately limits to unsupervised techniques. Thus it is our goal in this thesis to find a new method of obtaining data that may be used to inform these algorithms for better results.

(Haghighi & Klein 2006) describes a method of grammar induction with promising results on English using manually specified rules called prototypes. These prototypes provide means for a human to provide some minimal amount of language-specific knowledge that can capture important typological features, such as basic word order or phrase headedness.

Using these prototypes, we can place constraints on the inside-outside algorithm to both tie it to a set of pre-specified symbols to ensure induced rules remain linguistically meaningful, and only consider hypotheses known to be compatible with the language in question.

1.3 Interlinear Text

While grammar induction methods have been somewhat successful in finding productive grammars without annotated corpora, to produce a statistically stable and predictive model these methods often require large unannotated corpora. With many low-density languages, even unannotated data is hard to find. In cases such as these, unsupervised induction alone may produce poor results.

While seeking to minimize human supervision, including some level of linguistic knowledge could be both inexpensive and extremely beneficial. Fortunately, there exists data drawn from human annotators freely available on the web in the form of linguistic interlinear glossed text, or IGT. IGT is a method of presenting data designed to provide the reader of a linguistic paper with an annotated analysis of a foreign language. For examples and a further description, see Chapter 2.3.

Given that minimal supervision may be available in the form of IGT, even for languages without treebanks, we would like to incorporate this information using machine learning methods to produce an informed induction algorithm that should outperform purely uninformed methods.

1.4 Goals

While the construction of prototypes requires only a minimal amount of work from a language expert, interlinear information is electronically available for hundreds of languages in the ODIN database (Lewis 2006). Furthermore, it is feasible to extract prototypes from interlinear text automatically, as has been done recently by means of projection (Xia & Lewis 2007). Ideally, then, it should be possible to design a system that performs the end-to-end production of a PCFG for some low-density target language without human involvement. In this thesis we will present an attempt at such a system that is cross-linguistically motivated, with minimal cost.

This thesis contributes to the work on grammar induction by:

1. Suggesting a source of data previously unused in grammar induction methods

2. Proposing an extension to (Haghighi & Klein 2006) so as to automatically utilize this new data source for languages previously too data-poor for fully unsupervised methods

1.5 System Overview

The system designed in this thesis will be discussed as containing five main components, as shown in Figure 1.1. The first phase, selecting prototypes, was inspired by Haghighi & Klein's work described above, and is not a standard part of the inside-outside algorithm. We

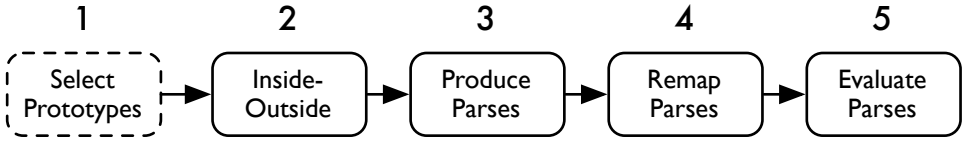


Figure 1.1: Overview of the steps involved in running the system end-to-end. Step 1 is the addition to standard implementations that we will be investigating.

will implement such a modification here, though leave it optional. The contribution made in this thesis lies in the new use of IGT in this step to inform the inside-outside algorithm. The details of extracting prototypes from IGT will be covered in Section 4.4.2.

The Inside-Outside algorithm indicated by step 2 lies at the core of the system. In order to discuss the prototype modifications, it is necessary to first understand the Inside-Outside implementation. As such, we will discuss this step first, in Chapter 3.

Finally, steps 3–5 are all portions of the evaluation process. The evaluation metric we will use is different from standard Parseval in the way labeled brackets are scored, due to label remapping done in step 4. The necessity and methodology for this remapping will be discussed in Chapter 5.

Chapter 2

LITERATURE REVIEW

The literature on grammar induction has a relatively short history, with many mixed results. In attempting to replicate previous work, we have found that much of the difficulty in grammar induction can be attributed to the sheer number of parameters involved in such systems and deciding how to deal with them. We have made a concentrated effort to point out these variables when possible and describe possible methods for addressing them.

2.1 Grammar Induction Techniques

2.1.1 EM-Based

Among the first to discuss an unsupervised approach at grammar induction was (Carroll & Charniak 1992). In this first attempt, the inside-outside algorithm (Baker 1979) is used to re-estimate grammar rules. The authors point out that due to EM's hill-climbing nature, a model with as many parameters as a good context-free grammar will have many local optima, making the initialization of the model important. Noting that lexicalized systems are possible, Carrol & Charniak instead bypass issues of ambiguity with word meaning that arise from such lexicalized systems and instead use POS tags as training tokens. This is a common approach throughout the literature, as such an approach addresses both issues with data sparsity that could be caused by lexicalized systems, and allows for closer scrutiny of the induction methods themselves by removing POS tagging from a system's pipeline.

Smart Grammar Creation

Intuitively, to induce a rule-based grammar using EM re-estimation, one could posit all possible rules with uniform probability and let the EM algorithm prune rules that fall below a given threshold. Practically, however, this pruning would result in a set of rules that would grow exponentially with the number of children allowed. Key in Carrol &

Charniak's approach is the pruning of this space by initializing the grammar on a subset of the training corpus by counting only the observed part-of-speech sequences seen in the data as possible rewrites. Using a yield such as: *DT NN VB IN DT NN*, their method would only generate rules that agreed with the observed tag sequence. For instance, NP → DT VB would never be generated from this yield since *DT VB* never occurs in the training data. NP → DT NN would be allowed, as would PP → IN DT NN.

The ultimate result of this method is a grammar pruning preprocess that produces a grammar similar to what a single iteration of EM might accomplish, but without the overhead of performing any actual parsing of the yields. A downside of this method is that it generates grammar rules only from those yields which it has seen. Given a yield sequence in which two symbols have never before been adjacent, they will never be considered a possible constituent. Though a similar effect may happen by training on raw data using EM, rules not seen by the algorithm may be smoothed by some amount and kept rather than being eliminated altogether.

Unfortunately, even with this clever model initialization, EM is not guaranteed to find a model close to the "true" model, and the accuracy for Carrol & Charniak's approach is poor on test data. Numbers on precision and recall are not given explicitly in Carrol & Charniak's paper, but experiments aimed at reproducing such work were done in (Haghighi & Klein 2006).

Improving upon the previous paper, (Smith & Eisner 2006) implements an annealing approach wherein certain rules (long-distance dependencies) are penalized in early iterations, but as early iterations stabilize, these rules are penalized less, with the hope that they will be incorporated in the new estimate if they are indeed necessary.

2.1.2 Distributional Induction

In contrast to the iteratively refining methods of the above systems, other work has focused on systems that conceptualize constituency not as sets of production rules, but rather in terms of mutual information found in the surrounding context of a word or phrase.

Such systems are generally quite successful at inducing classes of similar words and

phrases that occur in similar contexts. The difficulty with such systems, however, is finding a way to find what syntactic label a given constituent class represents, as the number of clusters to find is not necessarily best tied to the number of syntactic categories in a language. For instance, infinitival verb phrases have distributions unlike many other verb phrases, but are both labelled "VP" in the Penn Treebank.

(Clark 2001) describes a clustering system such as this. An unlexicalized system, Clark's uses an unbracketed POS corpora, and analyzes substrings of sentences to find similar context distributions. Clark then groups these substrings into clusters that are associated with syntactic categories. In combination with a mutual information metric to isolate true constituents from phrases that are simply co-occurring and minimum description length calculation to guide the grammar from one-rule-per-sentence, Clark's system is able to not only induce word classes, but possible PCFG rules for the grammar.

(Klein & Manning 2002) also describes a distributional-based system called a Constituent-Context Model (CCM). While operating primarily on distributional cues, Klein & Manning avoid the problem of finding non-constituent sub-clusters by restricting their algorithm to only consider sentence bracketings which are equivalent to parse trees; that is, bracketings that do not contain crossing brackets.

Both systems are quite robust at inducing constituency compatible with PCFGs, however, the grammars induced by both systems, when fully unsupervised, create automatically generated nonterminals, and without human intervention, the resultant PCFGs use arbitrary symbols to define their grammars. Ideally, we would like to both identify and label constituents with meaningful tags, and so we also investigate techniques where the algorithms are not entirely unsupervised.

2.2 Partial-Information Approaches

While EM iteratively refines a grammar to increase probability (and decrease entropy), it is well-known that it does not necessarily produce a grammar resembling those produced by supervised extraction from a treebank. Furthermore, there are many cases where a target language does not have a treebank large enough to extract a sufficiently sized supervised grammar, but does contain some amount of bracketing data.

2.2.1 Partial Bracketing

(Pereira & Schabes 1992) investigates a method of constraining the Inside-Outside algorithm to obey restrictions on bracketing using bracketings extracted from a small section of treebank material. Working on the ATIS corpus with bracketing extracted from a partially overlapping section of the Penn Treebank, this system is modified from standard implementations of the inside-outside algorithm so that, when presented with a string from training data, only bracketings which are compatible with known extracted parses are used.

This modification to the inside-outside algorithm is similar to the one used in the system we present here, and details on the modification can be found in Chapter 3.7.

2.2.2 Transformation-Based Learning

In a departure from EM-based approaches, (Brill 1993) devises a system that, while iteratively refining, does not begin with a hypothesized PCFG, but rather begins with a naïve bracketer, such as a strictly right-branching parser.[1] After starting with such a baseline system, TBL uses a small training corpus of bracketed sentences to propose "transformations" to the naïve bracketing, which include moving or deleting brackets to new positions.

Possible transformations are enumerated, applied to the naïvely-parsed data, then the transformation which increases the performance of the parses relative to the bracketed corpus is incorporated into the system.

Unfortunately, though this system is quite efficient at inducing bracketings, those that it produces are in fact unlabeled. Labeling constituent bracketings is quite difficult, and so our approach attempts to find a method which can perform both bracketing and labeling.

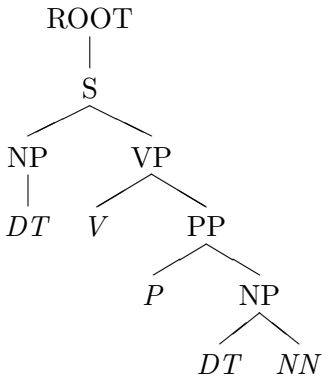
2.2.3 Prototype-Driven Approach

(Haghighi & Klein 2006) describes a method of injecting constraints into an unsupervised grammar induction system by way of specifying syntactic "prototypes."

---

[1]Though Brill gives right-branching parsing as an example of a naïve approach, as Haghighi & Klein point out, English is in fact a strongly right-branching language. Thus such a parser captures a significant fact about English and is not truly "naïve."

The prototypes described by Haghighi & Klein closely resemble context-free grammar rules, but rather than describing nonterminal nodes in terms of preterminals and nonterminals, they describe a nonterminal in terms of the sequence of leaf nodes, or yield, that they dominate. Figure 2.1 is an illustrated example of a prototype yield compared to a standard CFG rule.



CFG Rules:
S → NP VP
VP → *V* PP
PP → *P* NP

Prototype Yields:
S → *DT V P DT NN*
VP → *V P DT NN*
PP → *P DT NN*

Figure 2.1: CFG Rules vs. Prototype Yields

This method of supplying supervision is attractive for several reasons. First, obtaining prototypes from a linguistic informant is vastly less time consuming than training annotators. Secondly, for many languages with small speaker populations, finding annotators may be difficult or impossible, and defining prototypes is a task that could be performed by a linguist with a copy of a grammar for the language, or perhaps even a native speaker with limited linguistic training. Finally, revising annotations of hundreds or even thousands of trees would be an intensive undertaking, while a list of approximately twenty prototypes could be modified in a matter of minutes. In this thesis, we have concentrated on extracting such prototypes from IGT data, and a presentation of the form and use of these prototypes can be seen in Section 4.4.

Haghighi & Klein's results are impressive for a nearly unsupervised algorithm, producing labelled F_1 scores of around 0.65 in English. The labeled score for Chinese was 0.39, with a best-in-class unlabeled F_1 performance of 0.532, though the prototypes for this language were extracted from a treebank rather than manually specified. This success suggests that,

(1) Rhoddod yr athro lyfr i'r bachgen ddoe
gave-3sg the teacher book to-the boy yesterday
"The teacher gave a book to the boy yesterday"
(Bailyn, 2001)

Figure 2.2: Example IGT

if valid prototypes may be extracted from interlinear sources, that prototypes could greatly improve parsing performance.

2.3 Interlinear Glossed Text

Interlinear Glossed Text, or **IGT** is a form of presenting data commonly used in linguistic papers to highlight features of a language. These citations are found in many linguistic papers, and often, though not always, follow de-facto conventions for annotation. These small snippets can hold a great deal of linguistic information about a language.

A typical example of IGT might look something like the Welsh example in Figure 2.2. Looking closely, this example contains not only a parallel sentence in Welsh and English, but also a middle gloss line which uses the same words as the translation line, but in Welsh order. (Xia & Lewis 2007) notes that with some clever techniques, this gloss line actually contains implicit alignment information between the source sentence and the English translation.

Furthermore, since the translation line is in English, though no syntactic information is provided, high-performance parsers such as Charniak & Johnson's reranking parser (Charniak & Johnson 2005) can be used to provide such syntax.

Of course, a parser such as this does not often exist for languages that IGT is often used for. Xia & Lewis propose, however, that syntactic information can be obtained for the source language using this English parse and structural projection (Yarowsky, Ngai & Wicentowski 2001) onto the source line.

2.3.1 Structural Projection

Taking the IGT example from Figure 2.2 as a source, Figure 2.3 illustrates how alignment data can be inferred from the gloss and language lines. Using a parser on the English

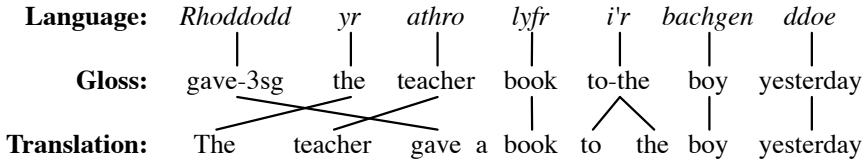


Figure 2.3: Implicit alignment information in an instance of IGT

translation yields the parse tree found in Figure 2.4(a).

If the alignments determined from the IGT are used to align the English words with those from the language line, these foreign words can be inserted in place of the English words. Unaligned English words, such as the article *a* are removed. Foreign words aligning to multiple English words are duplicated, as with *"i'r"*. Figure 2.4(b) shows the tree that results from this step.

Finally, after replacing the English words with those from the language line, the nodes must be reordered. Each node's children are reordered with the following constraints:

1. Children whose source spans do not overlap are ordered with respect to their source position.

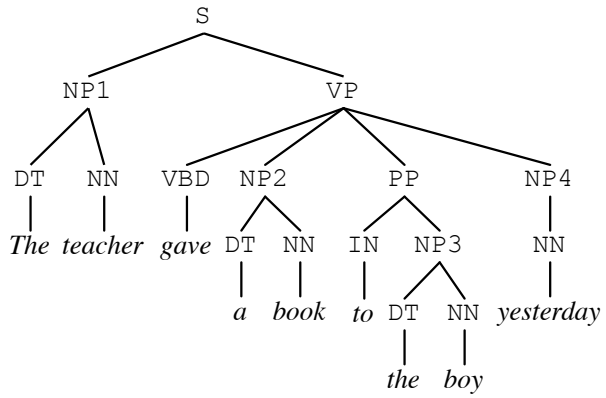
2. If a child A's span is a strict subset of another child B, it is removed and its children are promoted; that is, they become children of B.

3. If two children's spans overlap but are not strict subsets, both are removed and their children promoted. If they are leaf nodes with the same spans, they are merged.

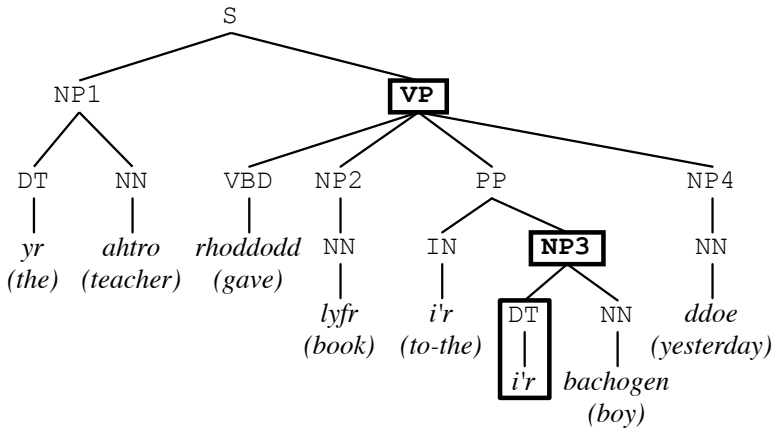
The boxes in Figure 2.4(b) show the nodes that will be removed by this process. The end result is the tree seen in Figure 2.4(c).

2.3.2 Difficulties with IGT Projection

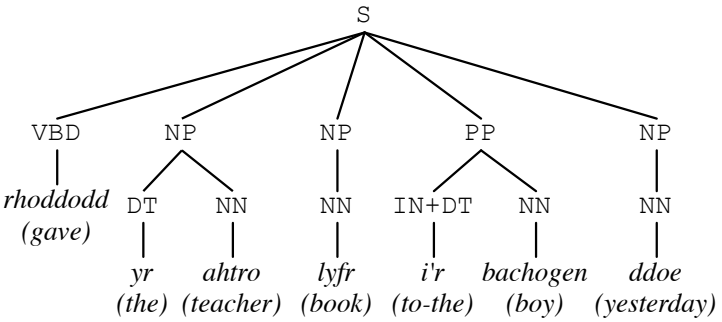
Though the projection algorithm given above can give us a great deal of information, the above example can be somewhat idealized to a proper digital format, adhering to a generally



(a) English phrase structure



(b) English phrase structure projected onto source



(c) Source phrase structure after processing

Figure 2.4: Demonstration of projection algorithm, as implemented in (Xia & Lewis 2007)

standard form of IGT. ODIN, the Online Database of INterlinear text (Lewis 2006), has a large repository of interlinear text, much of which has been gathered via crawling linguistic publications on the web. Improving the process of finding and detecting interlinear text in such papers is the subject of ongoing work, but instances are sometimes corrupt, or use unusual conventions for annotation. Furthermore, for non-Latin orthographies, conversion from postscript to other formats can be problematic.

Additionally, the nature of IGT projection leads to two forms of bias. The first is a bias towards English structures, given that the projection algorithm relies on using English parse trees for projection. While this is certainly a concern, (Lewis & Xia 2008) has shown that some basic typological features, such as word order, can be extracted from languages with little relation to English, such as Japanese and Korean with relatively high accuracy given enough instances of IGT.

The second form of bias is due to IGT itself, which tends to be used by linguists for citing selective examples within a language, and might show a bias toward nonstandard or outlying usages, depending on the intent of the author. The results of previous work with IGT, however, suggests that given a large enough variety of authors, rather than showing selective bias, quite a broad range of a language structure can be recovered, and this bias is less of a concern than one would tend to believe.

2.3.3 Using IGT

Using structural projection in combination with IGT, we now have a jumping-off point for many syntax-related tasks on a great number of languages. The ability to induce typological features such as word order suggests that there is a possibility of harnessing this data to find simple, prototypical examples of a language's structure.

2.4 Summary

Researching previous grammar induction methods has proven to be useful, as there are many methods available, each coming with unique benefits and pitfalls. In the end, we found that a prototype-modified EM approach was well-suited to the type of information that can be

extracted from IGT instances, as well as being conceptually simple. The following section will discuss this approach, as well as the prototype modification.

Chapter 3

THE INSIDE-OUTSIDE ALGORITHM

The Inside-Outside algorithm is a well-known member of the class of EM algorithms designed for use on phrase structure trees. Though the name "inside-outside" has been used in this paper for clarity thus far, we will refer to it more commonly as simply "EM" in later chapters. While the standard implementation is common and used as a baseline in our results, we include this chapter in order to provide clarification for the prototype modification.

3.1 Overview

Grammar induction techniques are designed to find patterns among large corpora of unannotated text. Clustering techniques can find commonly co-occurring tokens to determine constituency, but with no supervision it is difficult to determine labels for these constituents. In the worst case, such an algorithm may determine that n words fall into n constituent

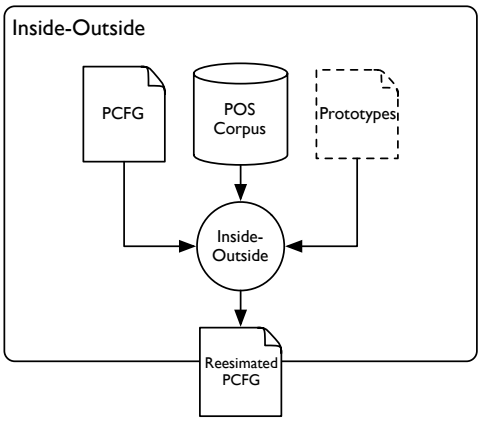


Figure 3.1: Illustration of the inputs and output of the Inside-Outside algorithm. The addition of a prototype file is a non-standard modification made in (Haghighi & Klein 2006), which we follow in our implementation.

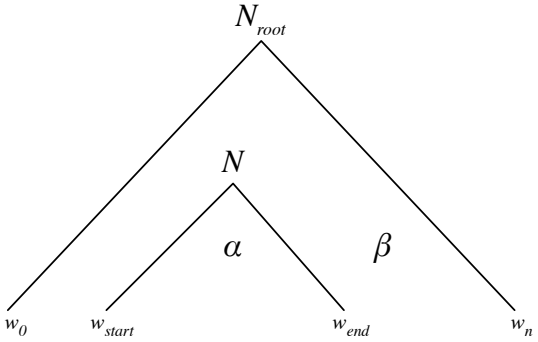


Figure 3.2: Visualization of inside and outside probabilities.

types.

The inside-outside algorithm, on the other hand, can be given a PCFG as input, and seeks merely to re-estimate the probabilities of the rules of that grammar in relation to a training corpus, and does not generate new labels. While this constraint on labelings enables the algorithm to limit the search space, it also means that we have a known tagset. The choice of a tagset will be discussed in Chapter 4.6.

The training corpora can consist of words, or just their part-of-speech tags. As has been done with previous work on grammar induction, we will leave the part-of-speech tagging task aside for the purposes of this system to avoid the complications introduced by word sense ambiguity and the data sparsity problems that may happen when words as tokens.

The inside-outside algorithm is an iterative, hill-climbing algorithm, named for the fact that its calculations are based on the inside and outside probabilities of strings in its training corpus. The inside probability α of any labeled subtree is the probability that its nonterminal label N may dominate a yield w_{start} to w_{end}. The outside probability β is the probability that the given subtree could be contained in the context it is found. Figure 3.2 gives a visualization of these probabilities.

These probabilities used by the inside-outside algorithm are derived from a PCFG and its production rules. Such a grammar must be given to the algorithm as input, along with a corpus of yields to analyze. An illustration of the system implemented here is given in

Figure 3.1.

3.2 Dynamic Programming Implementation

Although the inside-outside algorithm is relatively straightforward, it is only efficient when implemented using a dynamic programming technique that makes use of charts to hold the inside and outside probabilities. Appendix B gives a pseudocode reduction of the algorithm that will be referred to in this section.

3.2.1 Variables & Terms

To simplify the pseudocode, some assumptions concerning the variables are made. Rules, for instance, are considered here to be binary. Binarization can be done on the fly, but complicates the code and has been left out, as the grammars used in this experiment do not need such binarization. Such binary rules have a general form:

NP → DT NN

lhs → ***rhs****(symbol0, symbol1)*

Thus, *rule.rhs*[0] is shorthand for the first symbol on the **r**ight-**h**and-**s**ide of the binary rule.

3.2.2 Charts & Fill Order

"Charts" are the backbone of many dynamic-programming techniques; in this case, a chart as shown in Figure 3.3 happens to coincide quite well with a way of conceiving syntactic trees. In this Figure, a hypothetical chart is shown that correlates with the parse tree to the right. The dark shaded chart cell represents the cell currently being examined, spanning the tokens of the input sentence from positions 3 to 5. This span corresponds with the syntactic tree's NP node dominating *the dog*.

The algorithm consists of two charts, representing two sets of distributions. The inside chart's cells store the probabilities of the dominated string being formed by a given nonterminal, and is the product of all the dominated nodes beneath it. As such, the inside chart must be filled bottom-up.

The outside chart's cells represent the probabilities of a symbol occurring in the given

context of a parse. To fill the outside chart, we must know all the possible ways that the tree could be formed around the node being examined, so the inside chart is used in calculating the outside probabilities.

Finally, the combination of inside and outside probabilities represents the probability of a node yielding a certain string in the context of that position in the tree. The concept represented by this probability is important, as restricting the possibilities considered by the algorithm here is the key to constraining grammar induction to a more felicitous search.

3.2.3 Lookup Optimization

Additionally, though it has been greatly simplified here, the lookup of rules given in the pseudocode in Appendix B as simply (*rules* where...) is best accomplished by using hashes or arrays that index the rule by its symbols, so when filling chart cells, only rules that could possibly fit the current cells are looked up.

3.3 Walkthrough

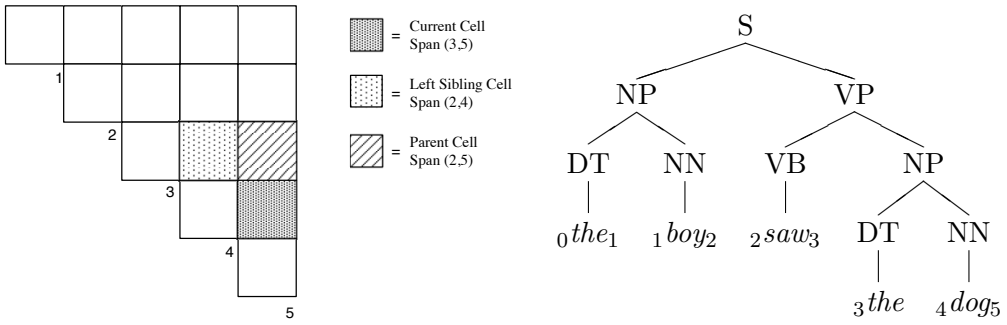
The algorithm can be divided into three main sections:

1. "Seeding" the inside chart
2. Filling the inside chart
3. Filling the outside chart

3.3.1 Seeding the inside chart

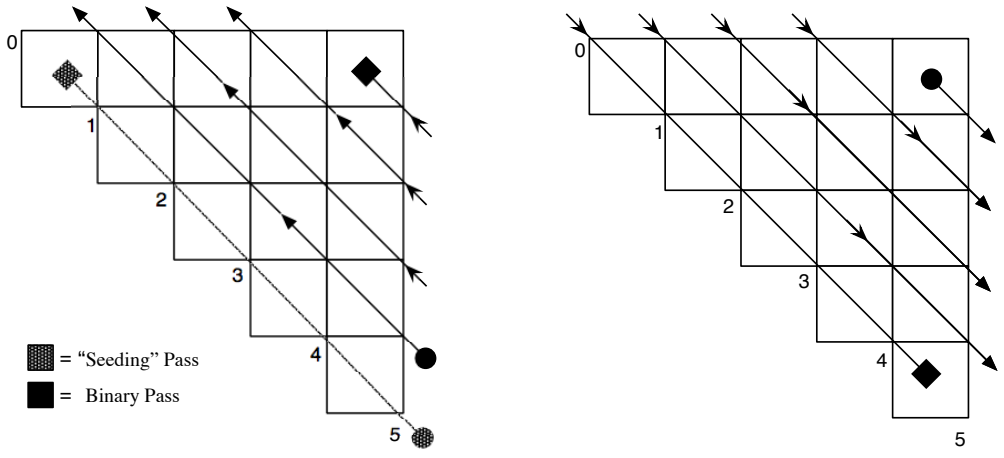
Lines 1–7[1] deal with setting the values for the chart cells that represent the POS tags assigned to the terminals in the sentence. In lexicalized implementations, ambiguity between words and POS tags can be introduced here. The implementation used here, however, is unlexicalized, so the seeding here consists of a 1:1 relationship between "dummy" terminals (DT_t) and standard POS preterminals (DT).

[1]For line numbers, please refer to the pseudocode in Appendix B.



0 The 1 boy 2 saw 3 the 4 dog 5
0 DT 1 NN 2 VB 3 DT 4 NN 5

Figure 3.3: Visualization of a chart, and the syntactic tree it represents



(a) Order for filling inside chart, bottom-up

(b) Order for filling outside chart, top-down

Figure 3.4: Order for filling the inside and outside charts

Since these preterminal to terminal expansions should be unary, this step is generally performed separately, as can be seen by the fill orders in Figure 3.4(a).

3.3.2 Inside Chart

Lines 8–27 of the code cover the process of filling the inside chart cells. The variables *start* and *end* refer to the entire span dominated by this node, while *split* is a moving split point that is used to try every possible binary division of children for the node. The fill order for this chart can be seen by the illustration in Figure 3.4(a).

Every rule that could expand to symbols with nonzero values in the child cells is considered a possible rule for the current cell, and the inside probability is the joint probability of both left and right symbols occurring as defined by the rule and the rule probability itself. This probability is set on line 20.

If, after filling the inside chart, the root node does not contain any symbol for which a start symbol expansion is defined, the parse fails and the sentence cannot be used to calculate revised rule counts. This is the primary reason we would like to specify all possible rules at the beginning.

3.3.3 Outside Chart

Lines 28–85 cover the filling of the outside chart, which, given a successfully filled inside chart, proceeds recursively from the root node down. This fill order can be seen in Figure 3.4(b).

Of special note is line 53, where the expected counts for every encountered rule are incremented with the combination of inside and outside probabilities. The accumulation of these counts is the implicit E-step in the algorithm. At the completion of the EM pass, the set of pseudo-counts will be normalized into PCFG compatible distributions and used as the grammar for the following iteration.

3.4 Code Package

The implementation of the algorithm used in this thesis is a modified version of Mark Johnson's Inside-Outside package, available at `http://www.cog.brown.edu/~mj/Software.htm` (Johnson n.d.). Aside from the usual algorithm illustrated in the code in Appendix B, Johnson has made two significant modifications:

- Internal Binarization of rules with three or more right-hand symbols
- Unary rules are allowed, with multiple application by precomputing a unary closure.

While these modifications are useful for parsers, our grammar is strictly binary with the exception of *preterminal* $\rightarrow$ *terminal* and *start symbol* $\rightarrow$ *nonterminal* rules, which are part of standard implementations.

3.5 Output

At the end of each iteration of the algorithm, the array *expectedCounts* is produced. This array is, in essence, a PCFG representation with newly re-estimated weights. By normalizing those weights to probability distributions, it is easily converted to the normalized PCFG format we desire to produce parses.

Once we have this re-estimated PCFG it can be fed into a CYK parser along with the raw text, or, as in our implementation, a POS-tagged corpus, to produce parses. Additionally, the PCFG can be fed back into the algorithm for further iterations in hope of converging on an even better-performing grammar.

3.6 Stop Point

With each iteration of the inside-outside algorithm, the grammar is modified slightly so as to increase the likelihood of the observed data. As previously noted, the inside probability of a node represents the probability of a given string being dominated by a particular symbol. If our parses are rooted with a ROOT symbol, we can find the log likelihood of the data given our current grammar simply by summing the negative log of each string's inside probability.

It is common practice to use the delta between these likelihoods from iteration to iteration to determine when convergence has been reached. Again, however, as the likelihood of the data given our grammar is not the optimal method to gauge our grammar's performance, we may want to stop the iterations at an empirically determined stop point, rather than letting the algorithm run until convergence.

Section 6.4.3 covers experiments that we ran covering precisely this, the results of which can be seen in the graphs in Appendix E.

3.7 Prototype Modification

The complicated parameter space of the inside-outside algorithm contains many local optima that its iterative hill-climbing method may decide upon. While early research using this algorithm focused on the importance of finding an optimal starting point so as to get closer to the global maximum (Carroll & Charniak 1992), recent work has contemplated the possibility of instead limiting what hills the algorithm is allowed to climb.

In this thesis we desire to constrain the actual search of the algorithm. When unconstrained, the algorithm is free to posit probability weight for any possible grammar rule, and since our start grammar usually involves every possible rule, this lack of constraint can lead to some extremely poor choices.

Instead, since we plan to gather partial supervision, we wish to follow the example given by (Pereira & Schabes 1992) and use a small amount of supervision to constrain the choices made by the algorithm. Unlike the partial bracketings used in that paper, however, we look to the work done by (Haghighi & Klein 2006) using prototypes. These easily constructed examples are not only intuitive, they can be thought of themselves as partial bracketings. A further description of prototypes can be found in Section 4.4.

3.7.1 Constraining the Algorithm

As mentioned in Section 3.2.2, the key to constraining the algorithm is in constraining the inside-outside probabilities. Using prototypes, this process is actually quite simple. As the inside chart is built, the full span of each cell is known, and thus, the sequence of leaf nodes, or yield.

1: $yield_{ij} \Leftarrow (w_i \dots w_j$ from $sent$ where $i = start$ **and** $j = end)$
2: **if** $(\exists\ prototype_x : yield_{ij} = prototype_x.rhs)$ **and** $(rule.lhs \neq prototype_x.lhs)$ **then**
3: $\quad prob_{inside} \Leftarrow 0$
4: **end if**

Figure 3.5: Prototype constraint modification pseudocode

1: $yield_{ij} \Leftarrow (w_i \dots w_j$ from $sent$ where $i = start$ **and** $j = end)$
2: **if** $(\exists\ prototype_x : yield_{ij} = prototype_x.rhs)$ **and** $(rule.lhs \neq prototype_x.lhs)$ **then**
3: $\quad prob_{inside} \Leftarrow prob_{inside} \times \frac{0.4}{|rules|}$
4: **else**
5: $\quad prob_{inside} \Leftarrow prob_{inside} \times 0.6$
6: **end if**

Figure 3.6: Soft prototype constraint modification pseudocode

If, then, we have the prototype: **NP** → DT NN, and, as in the example in Figure 3.3, we are examining a cell spanning (3,5) which has the POS yield DT NN, we could force the algorithm to posit *only* NP as a possible label for that node. An illustration is provided in Figure 3.7.

Specifically, the lines in Figure 3.5 could be inserted between lines 20 and 21 in Appendix B. This simple modification states that any yield for which a matching prototype is found cannot use a rule that does not share the same left-hand label as the prototype.

While intuitive, this method can lead to difficulties. In Figure 3.8, a classic example of syntactic ambiguity is given. If we use the prototype: **NP** → DT NN IN DT NN – the yield dominated by the bolded NP in the left tree, when we are analyzing the tree on the right, we will falsely presume the span (2,7) to be a constituent headed by NP, when it is not.

As observed in (Haghighi & Klein 2006), this constraint is likely too strict. As an alternative to the strict constraint above, we follow Haghighi & Klein's example and implement the possibility of using a softer constraint for prototypes we are less certain of. Rather than the code in Figure 3.5, we suggest the modification in Figure 3.6.

This modification has the effect of shifting 0.6 of the probability mass for possible rules to the nonterminal specified by the prototype, and splitting the remaining 0.4 among the

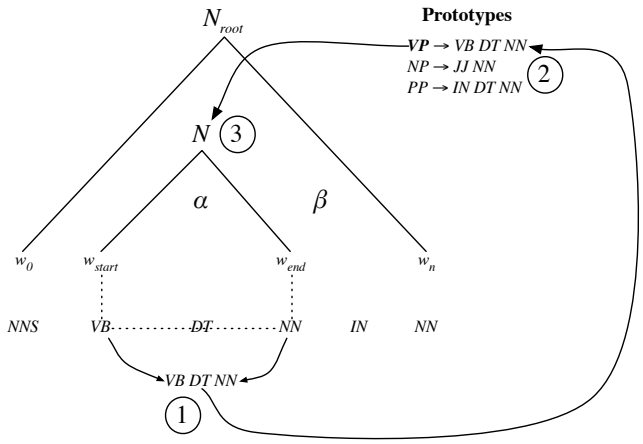


Figure 3.7: Illustration of inside-outside modification for prototypes.

remaining rules. In this way, we ensure that a proposed prototype that predicts a label inconsistent with any possible parse for the sentence will not result in a failed parse, but instead non-prototype rules will be used as backoff.

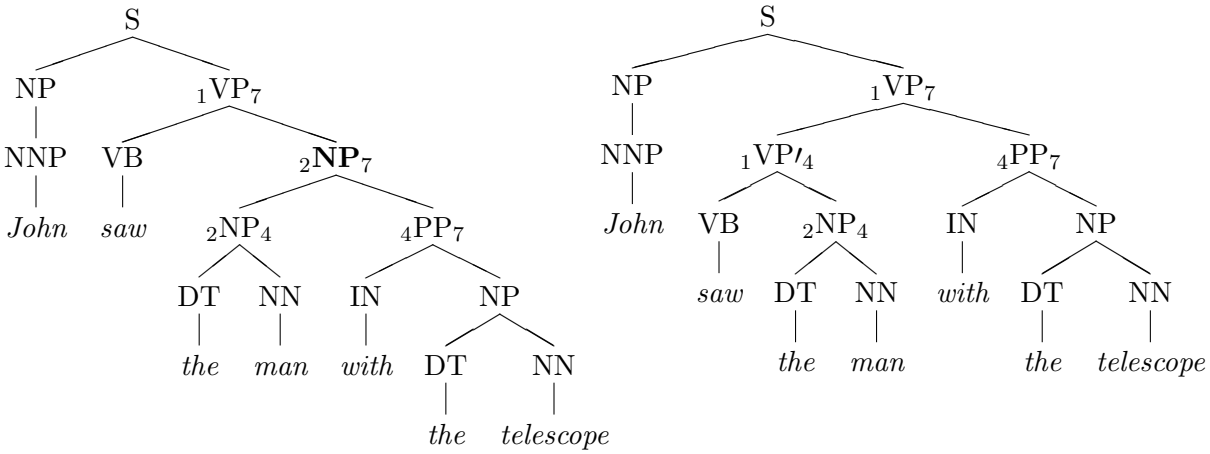


Figure 3.8: A classical example of syntactic ambiguity

Chapter 4

METHODOLOGY

As previously stated, our system centers on the use of IGT to extract prototype information that can be used to constrain the EM algorithm. In order to accomplish this, we must project syntactic information from the English translations found in IGT instances, extract this projected syntax into useful prototypes, and bolster these extracted prototypes by finding similar extensions to those already defined. We will provide detail in this chapter on how this system fits together.

Furthermore, one of the great difficulties in grammar induction is the number of variables, not only in the grammar itself, but the choice of language, the corpus provided for that language, even the symbols used within the corpus. Before addressing the details of the system, we will begin with a discussion of the corpora we have selected.

4.1 Corpus Selection & Preparation

4.1.1 Choice of Languages

Though the ultimate target domain of this project is languages for which no treebanks exist, some supervised data was needed to evaluate performance, at least under best-case scenarios. Furthermore, though data for over 700 languages can be found in ODIN(Lewis 2006), many languages have a mere handful of IGT instances.

We chose German for our experiments for several reasons. First, German possesses several treebanks. Treebanks would be required for evaluating our system. Second, the NEGRA (Skut, Krenn, Brants & Uszkoreit 1997) newswire corpus has been used in previous grammar induction attempts, and is even available in a non-crossing, Penn-Treebank-like format which could be easily utilized by our existing code. Finally, ODIN contains a substantial amount of IGT data for German—around 1,200 usable instances at the time of this writing.

	NEGRA10	WSJ10
Sentences	6,504	7,422
Tokens	42,331	52,248
Syntactic Tags	25	22
POS Tags	35	51

Table 4.1: Breakdown of NEGRA10 and WSJ10 statistics

(Klein 2005) also reports numbers on LDC's Chinese Treebank (CTB). Because Chinese has a greater typological dissimilarity from English than German, we would also liked to have used this source; however, due to time constraints and the author's unfamiliarity with Chinese, it was determined to leave those tests for future work.

4.1.2 Data Set

Following (Klein 2005) and (Haghighi & Klein 2006), we have processed the Wall Street Journal section of the Penn Treebank along with the NEGRA corpus to remove certain elements and lengthy sentences. The resulting corpora will be referred to as the WSJ10 and NEGRA10. The "10" suffix refers to restricting the length of sentences from the corpora to ten or fewer tokens, counted after removing null (trace) elements and punctuation. The specifics on removal are given below.

4.1.3 Removal Process

Since null elements show no manifestation in the data, learning the movement rules they are intended to represent is not possible when only raw yields are given as input. As such, any tree node whose child is a trace, such as (*-T *NONE*), is removed from the tree. Any node whose only child is removed by such a deletion is also removed.

Punctuation, while it may provide contextual cues to phrase structure, has been found in previous work to be not as strong a predictor as one may hope, and thus is also removed in the manner described above.

For the NEGRA corpus, the removed tags were the punctuation tags: {$ $. $*LRB* $*RRB*} as well as empty-element tags beginning with *.

4.1.4 Length Filtering

After null elements and punctuation have been removed, only sentences with ten or fewer words are kept, resulting in a total of 7,422 sentences for all Sections 0 through 24 of the WSJ corpus.

For the NEGRA corpus, (Klein 2005) reports a corpus size of 2,175 sentences. Since that publication, an updated version of the NEGRA corpus has been released with an increased amount of data. Following a similar process, we found that v2 of the NEGRA corpus consisted of 6,504 sentences.

4.1.5 Data Partitioning

It is generally standard practice when training a supervised system to segment data into training and holdout corpora. Such segmentation avoids over-fitting to training data, as rules extracted from a treebank may not always reflect performance on novel constructions.

As treebanks actually contain POS yields upon which the trees serve as annotation, such a corpus is actually available within the treebank itself. Furthermore, if we extract the POS yields only and no bracketing information, the system is unable to "cheat" by using training data that overlaps with test data. While this does not address concerns of over-fitting, since the system requires only POS yields for training, additional POS strings needing analysis can actually be added to the training data so that they can be accounted for.

Due to the desire to bypass the Part-of-Speech tagging problem, we have elected to use the part-of-speech tags from the treebanks we will be evaluating against as a training corpus, though this is not a requirement[1]. Indeed, training on a much larger POS-tag corpus would likely yield far better results than our small, sub-10,000 sentence training sets.

4.2 Grammar Definition

The grammars used in this thesis are binary PCFGs, as is standard for inside-outside and CYK compatibility. The only modification is that while our system is unlexicalized, our

[1]Personal communication with Aria Haghighi indicated that a similar approach was taken in (Haghighi & Klein 2006).

tools are designed for lexicalized systems. In order to produce a compatible grammar for a lexicalized system, we utilize preterminals, which are usually POS tags, with the constraint that they must have only one rule each, to a matching POS terminal. Full details of the grammar are given in Appendix A.

4.3 Grammar Generation

Not every method of grammar induction requires a grammar be created beforehand, but in the case of EM, an input PCFG grammar is required. There are a number of ways an initial grammar might be created.

1. Use CFG rules extracted from a bracketed source.

2. Generate possible rules from unlabeled data.

3. Generate every possible compatible rule.

The first approach is a standard practice for treebanks, using simple counts of rules occurring in the treebank to infer the probabilities for an PCFG. Though we are using treebanks for evaluation in our system, they are meant to be used only for proof-of-concept and our intent is to use IGT as an alternate source of supervision. While it is possible to extract PCFG rules directly from the projected IGT parse trees described in Section 2.3.1, such a grammar would not only be of low quality due to the noise inherent in these IGT-projected parses (see Section 2.3.2), it would more damagingly be extremely sparse. Such a grammar, when encountering unseen data, would likely be unable to parse and thus unable to revise the current model to fit the data. Due to these issues, we have not pursued option 1 for creating the grammar.

The second option is to "smart seed" the grammar, as done in (Carroll & Charniak 1992) and discussed in 2.1.1. Though such a method indeed reduces the grammar size, we found that doing so did not substantively help, as such a technique largely replicated the first iteration of EM when the grammar was restricted to binary, as in this thesis.

$$a_i = \frac{\frac{1}{N_A} + r_i}{1 + \sum_i r_i}$$

Figure 4.1: The formula for determining weight for the ith rule for nonterminal A, a_i, where N_A is the number of rules with A as their left-hand side. r_i is the noise generated for this rule where $0 \leq r_i < t$.

The third option of generating every possible grammar rule, though computationally expensive, is generally the path chosen for EM based methods, and is the one used here. This is also the primary motivator in limiting the grammar to binary rules, as allowing larger rules would grow the grammar exponentially, and both time and hardware constraints were a concern with following this method.

4.3.1 Random Noise Addition

It is common practice when using EM re-estimation to inject a small amount of noise to the initial grammars. This noise is intended to break the symmetry that may be caused by equally-weighted production rules.

In this thesis, such randomization is accomplished by using a random method to generate a pseudorandom number x where $0.0 \leq x < 1.0$. A threshold t is specified, and multiplied with x to adjust the amount of noise to within the given threshold. Randomization is applied to each production rule independently, and productions for each nonterminal are then normalized to sum to 1. An equation for determining the weight for a rule is given in Figure 4.1.

4.4 Using Prototypes

In order to use prototypes in the EM algorithm, we implement the modification laid out in Section 3.7 of using the nonterminal specified in a given prototype to override other options the algorithm may be considering when filling the inside chart cell.

The concept behind these prototypes is to use them to inform the algorithm. Where we get this information is another question. As Figure 4.2 illustrates, we actually have several

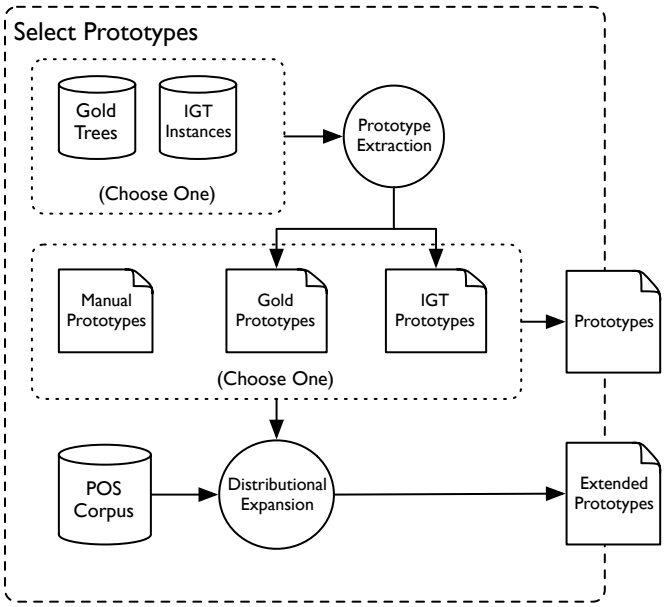


Figure 4.2: An illustration of the process of selecting prototypes for our system. Note that the prototypes can come from multiple sources, and each type of prototype may be extended with distributional induction.

choices for obtaining prototypes:

- Manually Specified (e.g. by a linguist or native speaker)
- IGT-Extracted (finding yields predictive of the projected parses)
- Treebank-Extracted (finding yields predictive of the treebank parses)
- Use no prototypes

Naturally, the last two cases are not truly intended for this system, but serve as convenient methods for determining an upper bound and baseline, respectively.

Furthermore, no matter how we obtain our prototypes, the prototypes can be extended by distributional clustering, so as to find prototypes such as *DT JJ NN* that may be found in a similar context to NP → DT NN, though not specified explicitly. First, we will address how prototypes are selected.

4.4.1 Manual Specification

Naturally, one of the most straightforward methods for obtaining prototypes is to have a linguistic informant or trained linguist (preferably a combination of the two) come up with a list of simple, short yields for each syntactic category in their language of expertise. (Haghighi & Klein 2006) use such an approach with English, and their results indicate that such a manually specified list was only marginally less successful than a list extracted from the treebank itself.

4.4.2 Extracting From Treebanks & IGT

While having a single linguist compile a prototype list is far less expensive than annotating a treebank, we would still like to find the feasibility of extracting prototypes automatically, so that parsing could be attempted automatically on hundreds of languages without human supervision.

If such a system is feasible, any language with adequate data in ODIN could have a basic syntactic parser prepared with no human interaction whatsoever.

Selection Criteria

The automatic selection of prototypes is a task that may be performed on any set of bracketed sub-trees. While we seek to use syntax trees projected from IGT data, this selection is a task that can also be performed on a treebank. We use the treebank as a way to demonstrate the selection criteria as well as provide an upper bound of sorts, using idealized data.

When selecting prototypes, there are two primary statistics that need notice. First, there is the simple count of yield occurrences for a given nonterminal, which can be thought of as the distribution $P(\psi \mid \mathrm{N})$, where N is a given syntactic label, and ψ is the terminal yield dominated by that nonterminal. This distribution captures the number of times a given symbol is associated with a certain yield. In (Haghighi & Klein 2006), this is the number maximized to cull a list of the top three yields for each nonterminal in the treebank. This "cheating" experiment yielded a very slight improvement over their manually-specified tagset in English.

	*VB*__	*IN*__
__*VB*	1	0
__*CC*	1	0
__*$END*	0	2

(a) Context array

VB ***DT*** ***NN*** *VB* *IN* ***DT*** ***NN***

VB ***DT*** ***NN*** *CC* *VB* *PDT* *IN* ***DT*** ***NN***

(b) Sample yields

Figure 4.3: Demonstration of counting contexts in yields

Though finding the optimal value of $P(\psi \mid \mathrm{N})$ appears to yield good results for extraction, it should be noted that the distribution over $P(\mathrm{N} \mid \psi)$ could also be used to find appropriate prototypes. This measure would give yields that are the most likely to be associated with a given nonterminal, and used to find prototypes with the lowest entropy, and thus the most predictive power. This distribution is useful in dealing with "overlapping" yields, that may be associated with multiple nonterminals, even if they are frequently occurring.

There are multiple ways to deal with such yields. As the induction algorithm used is probabilistic, it is conceivable to use any number of extracted prototypes and associate each with some confidence measure, our simple implementation uses a threshold of 0.9 for $P(\mathrm{N} \mid \psi)$ and throws out single occurrences found in the $P(\psi \mid \mathrm{N})$ paradigm.

4.4.3 Distributional Clustering

Prototypes created by any method of selection can be easily extended with distributional clustering on an unbracketed POS corpus. In this thesis, clustering is performed over the linearized treebanks, but could very well be done over any part-of-speech tagged corpus.

Context Vectors

Following (Haghighi & Klein 2006), clustering is done using context vectors that look one tag to the left and right of the yield in question. In this implementation, sparse vectors are created in the form of a two-dimensional matrix by examining the tags immediately to the left and right of each yield. If we let σ represent the left context and χ the right, then the contexts for *DT NN* are indexed by the ordered pair (σ, χ) in Figure 4.3(a). A special symbol is used for beginning and end of sentence boundaries. Context vectors are created

$$D_{KL}(P||Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}$$
$$D_{SKL}(P||Q) = D_{KL}(P||\gamma P + (1-\gamma)Q)$$

Figure 4.4: KL-Divergence vs. Skewed KL-Divergence

for every substring of terminals found in the corpus, including those yields defined by the specified prototypes.

Similarity Measure

After context vectors have been created, the vectors created from the prototype yields are compared with every other context using a skewed KL-divergence metric as a similarity measure, as defined in Figure 4.4. Following (Haghighi & Klein 2006), using a γ of 0.1, any context vector that falls within the threshold t of 0.75 is considered a suitable alternative prototype. If a substring does not fall within a skewed KL-divergence of 0.75 it is not considered to be reliably associated with a given nonterminal.

4.4.4 Automatic Weighting

When prototypes are found through distributional clustering, though they are likely good matches for previously defined prototypes, they do not fully override nodes the way the initially defined prototypes do. Instead, they are assigned 0.6 of the probability mass as described in 3.7.1. This allows other possible labels to be considered for this node in the event that the automatically induced prototype is wrong, while still giving it preference.

4.5 Working with IGT

Although IGT is a promising source for data, there is currently a bit of work needed to transform the current IGT instances pulled from ODIN to a usable form for prototype extraction.

```
(S1 (S (RB weil) (NP (PRP ich)) (VBP dürstet)))
```

Figure 4.5: Sample output of IGT projection. Note that the tags produced are Penn Treebank tags being used on German words.

4.5.1 Cleaning

Before instances can even begin to be used for projection, there is often a good deal of noise that must be cleaned from the instances. For example, while the original sentence is rarely an actual quotation, quotation marks are often used on the translation line to differentiate it from the gloss provided above. Numbering schemas, source citations, and other parenthetical explanations of linguistic phenomena are among the extraneous information found on both gloss and translation lines. These artifacts are removed with the use of simple regular expressions.

4.5.2 Nonstandard IGT

In addition, in papers where multiple instances are used consecutively for a similar phenomena or multiple interpretations of a similar phrase, or anywhere else the author finds them unnecessary, gloss or translation lines may be omitted. Since both of these are required for the projection algorithm, these instances are pruned from those to be projected.

4.5.3 Parsing The Translation Line

After cleaning, a parse of the translation line is performed using Charniak & Johnson's reranking parser (Charniak & Johnson 2005). As it was trained on the Penn Treebank, the output is formatted in PTB style brackets and uses the PTB tagset, as in Figure 4.5.

4.5.4 Word Alignment

Before the syntactic tree can be properly projected, alignment between the translation and gloss lines must be found. Currently, this is done by attempting to match the English words with those on the gloss lines. Since quite often the form of an English word may be changed in the gloss (e.g. `dog-PL` instead of `dogs`, or `give-PAST` in place of `gave`), a morpher is

used on both lines before a match is attempted. For each match, the alignment between position in the phrase is stored for reordering the projected tree, as in Figure 2.3.

4.5.5 Projection & Extraction

Projecting the English parse tree consists of three tasks, illustrated by the trees in Figure 2.4.

1. Replacing English words with source words – Figure 2.4(a)

2. Reordering source words in the tree based on stored alignments – Figure 2.4(b)

3. Reattaching unaligned source words and deleting / merging nodes as needed. – Figure 2.4(c)

The end product is these tasks is a parse tree for the target language, albeit with words tagged "unaligned" where alignment failed to place a word within a syntactic label. Extraction of the prototypes is finally performed following Section 4.4.2, with the exception that nodes must not contain any "unaligned" tokens to be considered for a prototype.

4.6 Tagsets

While the choice of a tagset is pre-defined in supervised systems, care must be taken when choosing a tagset for an unsupervised method.

4.6.1 Problems with EM

The choice of tagset is particularly important when dealing with EM algorithms, as the objective function of these algorithms is to optimize the likelihood of the data, *not* the robustness of the model.

As a result, the choice of tagset can have effects on the way the Inside-Outside algorithm treats syntactic categories. Remember that EM works by refining the model for ever-increasing *likelihood*. As the model becomes better fit to the data, its entropy decreases. This lower-entropy seeking method of iteration can cause rarely-occurring categories to be

favored, since a rule predicting a rare category would likely not be ambiguous, and thus have lower entropy. In this vein, the ideal grammar for EM would be one in which every symbol was unique, and had a single, 1.0 probability PCFG rule associated with it. As every string would be predictable and unambiguous, it would be easy to fit a model to such data.

Naturally, such a grammar is never the case, but should serve to illustrate how EM can do better with finer-grained tagsets. For instance, the tagset used in the NEGRA corpus makes a number of distinctions not made by the Penn Treebank tagset, such as using different labels for postpositions, adpositions, and circumfixes, whereas PTB only uses one symbol, IN, for prepositions. Although these German tags may deal with different constructions than English, there are certainly ambiguities introduced into English with such constructions as *he went out*.

Such specifics in a tagset make for easier induction as ambiguity is decreased. Using EM with the full NEGRA tagset and no prototype supervision, we find that the induced categories were quite close to those using supervision, though those unsupervised categories do not map to the proper symbols without guidance.

In general, we aim for a tagset that uses minimally eight syntactic categories:

NP	S	PP	ADVP
VP	QP	ADJP	MISC

Maintaining a grammar with these minimal symbols prevents some of the most important syntactic patterns from being collapsed and lost, while attempting to maintain cross-linguistic relevance.

4.6.2 Splitting Categories

As finer-grained part-of-speech tags can assist in grammar induction, so too can more finely grained syntactic categories. For instance, in PTB notation, an NP can represent DT NN, NNP NNP or NNS PP. As is expected with any syntactic category, any of these productions are largely interchangeable, but as has been discussed previously, the nature of EM leads toward grammars with less ambiguity. Thus, it may be wise to follow the lead shown in

(Klein & Manning 2003), where these NPs are given separate labels, NP_1, NP_2 ... NP_n. These NP sub-categories would be easier for the EM algorithm to identify, yet still be easily collapsed down the line.

4.6.3 Tagset Mapping

When it comes to cross-language projection done in this thesis using IGT, the choice of tagset gets more complicated. As described in Section 2.3.1, the prototypes that are ultimately produced by IGT projection have been projected from English, and thus are using the tagset used in parsing the English translations. In this thesis, that tagset is the WSJ10 tagset. Unfortunately, this results in prototypes whose labels and yields do not match that of the target language. In order for these prototypes to be of use, they must be able to be matched to the yields input to the inside-outside algorithm. For instance, we might extract the prototype VP → VB PP from IGT, but if the symbol VB is never seen in the training data—only VV—the prototype will never match. As a result, we must find some way to make these tagsets compatible.

Mapping Methods

To make these tagsets compatible, we have a decision to make: whether to change the English tags to foreign tags, or the foreign tags to English ones. While the former option would appear to be obvious, given that we ultimately want the foreign tagset, it is not always so easy.

As was noted previously, the German tagset has a good deal more POS tags than English. Thus, although many of the German tags represent similar constructions in English, we would have to systematically be able to separate equivalent tags to convert. This could be done perhaps with contextual rewrite rules (e.g. a trailing English preposition IN would be labelled instead with some postposition label), or we could simply use an English parser that produces finer-grained distinctions. Both methods would result in producing additional English tags that would ultimately make the two tagsets more readily comparable.

For simplicity, we have chosen to go the other route and map the German tagset onto

the English one. This means POS tags that are distinct in German become collapsed into a comparable English label. For instance, VVINF (Infinitive), VVIZU (Infinitive with *zu*), VVIMP (Imperative) and VVAMP (Imperative w/aux) all are collapsed to the single English label for base form, VB. The full tagset mapping table can be found in Appendix D.

Using this POS mapping means that the POS corpus to be used by EM must be remapped this way, and the grammar generated must use this tagset. Finally, the produced parses will use the English tagset.

Ultimately, for the low-density languages this thesis was intended for, this tagset mapping may not be undesirable. Many labels, such as VB tend to be fairly cross-linguistic, and even on a typologically dissimilar language, an English tagset may be able to give at least a basic picture of the target language's structure.

4.6.4 Evaluation Tagset

For the purposes of this thesis, we wanted to compare the parses produced by the system with a gold standard. As noted above, however, we chose to use a mapping for the German that resulted in English categories. Although our final parses end up with WSJ10 tags, however, our final evaluation is still run against the unmodified NEGRA10 treebank. The evaluation method discussed in Chapter 5 resolves this inconsistency by automatically matching the tags in the produced parses with those in NEGRA, so labeled scores are not penalized for this difference in tags.

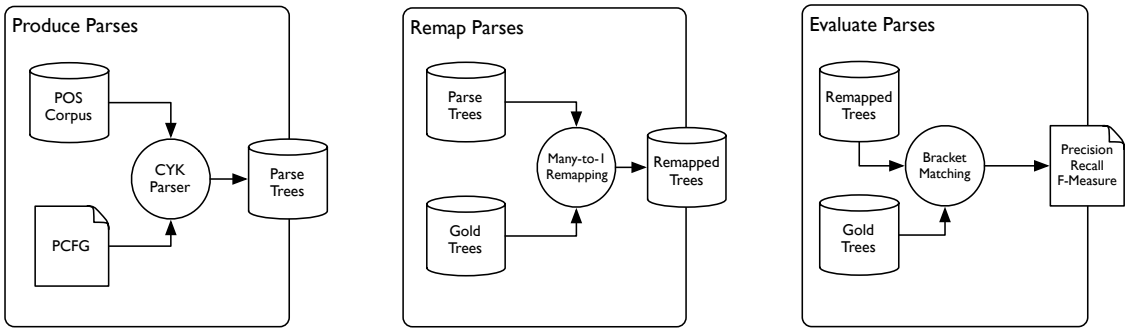
Chapter 5

EVALUATION PROCEDURE

Though bracket matching has been a rather standard process for evaluating PCFG parsing for some time, evaluation metrics for induced grammars can prove to be a bit trickier. With fully unsupervised systems such as those discussed in 2.1.2, though they may generate unlabeled bracketings, the labels used may not have any relevance to those used in the gold standard.

A strict, labeled Parseval score for such a system may very well produce a labeled F_1 close to 0 due to such problems. This does not mean, however, that such systems do not accurately group constituents into proper classes, only that they have difficulty labeling them. EM algorithms that begin with a PCFG grammar with the proper symbols may map symbols incorrectly, even if they induce proper structure.

For these reasons and for comparison with previously reported work, our primary eval-



(a) Parses are produced on the evaluation POS corpus by running the inside-outside produced PCFG through a CYK parser.

(b) Before scoring, the produced parses are remapped using a many-to-one mapping described in Section 5.2

(c) Finally, these remapped parses are evaluated against the gold standard using the bracket-matching described in Section 5.3

Figure 5.1: Illustration of the steps involved in the evaluation process.

uation metric will be a labelled bracket match utilizing a many-to-one mapping following (Klein 2005). The intention of this thesis, of course, is to produce labeled parses that have correctly induced classes *and* a proper labeling. As such, we will report these when applicable.

5.1 Producing Parses

Before we can even begin the bracket-matching process, we must create parses from the inside-outside produced PCFGs. This process, as illustrated in Figure 5.1(a), is accomplished by means of a CYK parser. For this thesis, Mark Johnson's implementation found at `http://www.cog.brown.edu/~mj/Software.htm` (Johnson n.d.) is used for its compatibility with the PCFGs produced by the EM package.

5.2 Many-to-One Mapping

In order to determine what syntactic categories in the gold standard the induced categories map to during evaluation, the trees in both the output parses and gold standard are scanned. If a node in the output has a span that matches that of a node in the gold standard (a standard unlabeled match), a count is taken of the output label and the label in the gold standard.

After making a table of all such counts, each output label is individually relabeled to the gold label which it matched most frequently. Multiple output symbols are allowed to be mapped to the same gold labels.

Again, the purpose of performing such a mapping is to gain insight onto how well purely unsupervised methods perform the task of finding syntactic categories; but it should be noted that when prototype supervision is used to guide the process, such a remapping may not be needed.

5.3 Bracket Matching

After the many-to-one mapping has been performed, bracket matching is done with two additional constraints, following (Klein 2005). The ROOT or S1 node that dominates every successfully parsed sentence is not counted in either the gold or output trees, as it is obtained

for free. Furthermore, any node that dominates a total of only one leaf node is not counted in either tree, as such spans are impossible to obtain using a strictly binary system.

5.4 Results Averaging

When running experiments for this thesis, all the grammars were initialized with a small amount of random noise to break symmetry. To ensure that results were representative, each experiment was run with 10 parallel instances, and the results averaged across the numbers. Individual variation between runs was usually within 2 points by convergence.

Chapter 6

EXPERIMENTS

6.1 Overview

In order to determine what contribution our IGT-informed system will make to previous grammar induction work, we ran experiments designed to cover a wide range of settings that might affect the induction algorithm. For our purposes, we wish to find outcomes for the following tests:

1. Naïve baselines (arbitrary branching)

2. Upper bounds on binary grammars

3. Uninformed EM performance

4. Replicate previous prototype system results (Haghighi & Klein 2006)

5. Upper bound on prototype performance

6. Performance of IGT-extracted prototypes

In addition to these primary goals, the EM algorithm itself has several settings rarely documented in the literature. We would also like to examine:

7. Levels of random noise added to initial grammar

8. Effects of changing number of grammar symbols

9. Optimal number of iterations

WSJ10 BASELINES

	Labeled			Unlabeled		
	Prec.	Rec.	F_1	Prec.	Rec.	F_1
LBRANCH	–	–	–	26.37	32.62	28.7
RBRANCH	–	–	–	55.16	70.04	61.7
UBOUND	78.75	98.48	87.52	78.75	100.0	88.11
EM	37.00	46.98	41.40	44.64	56.69	49.95

Table 6.1: Baseline results using naïve left-branching and right branching parsers. Upper bound is the limitation of binary grammars on non-binary gold standard. The upper bound on labeled precision is due to unary chains.

NEGRA10 BASELINES

	Labeled			Unlabeled		
	Prec.	Rec.	F_1	Prec.	Rec.	F_1
LBRANCH	–	–	–	26.37	47.56	33.93
RBRANCH	–	–	–	32.85	59.26	42.27
UBOUND	56.25	99.99	72.00	56.25	100.0	72.00
EM	31.92	57.57	41.07	43.54	78.53	56.02

Table 6.2: Baseline results for the NEGRA10 corpus: naïve left-branching and right-branching parses and binary-limited upper bound. EM result is uninformed inside-outside run with noise threshold of $t = 0.1$.

6.2 Baselines & Upper Bounds

6.2.1 Baselines

Naïve baselines, in the form of left- and right-branching bracketings are one of the first tests done. The results of these bracketings give an idea of baseline performance, but it should be reiterated that some languages, such as English, have a strong preference for direction of bracketing. A grammar induction system may learn significant language structure while still performing below the baseline. Our English systems fall below the right-branching baseline (RBRANCH) for English, a very high unlabeled F_1 of 0.617 but easily beat the left-branching baseline (LBRANCH) of 0.287 due to English's strong preference for right-branching structures. Note that while these naïve systems are able to create bracketings, there is no appropriate way to assign labels for the brackets, so only unlabeled scores are

given.

Our system for German, which has less of a strong preference for one direction of bracketing, easily beat both the left-branching baseline's F_1 of 0.3393 and right-branching 0.4227. The English results for these baselines can be found in Table 6.1, and those for German in Table 6.2.

Comparison with Previous Work

The scores we obtained from these baselines matched those reported in (Klein 2005) for WSJ10. Though our data set for NEGRA10 is slightly different from Klein's as noted in 4.1.2, our German baselines are very similar to those previously reported.

6.2.2 Upper Bounds

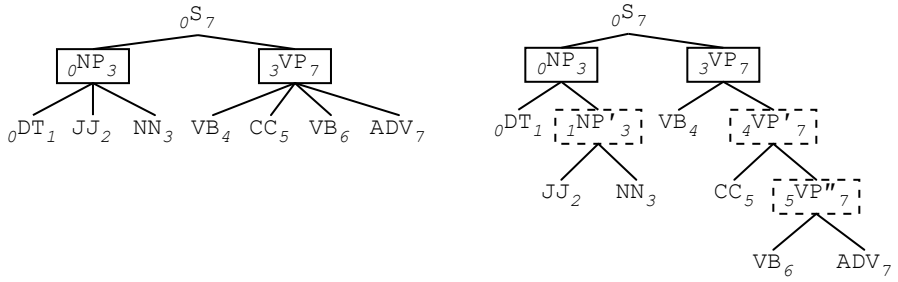
As noted in Section 4.2, the grammars used in this thesis, with the exception of terminal and start symbol rules, are strictly binary for performance reasons. When running such a grammar on treebanks that are "flatter," that is, treebanks where nodes are allowed to dominate more than two children, there is a limit to the performance binary grammars may achieve.

It is possible to binarize a ternary production by converting it to a series of binary productions, but doing so will penalize precision due to hypothesizing nodes that do not occur in the gold standard.

As can be seen in Figures 6.1(a) and 6.1(b), the solid boxed spans in the flat tree can be matched by equivalent spans in the binary tree, but the dashed spans are spans generated by binarization that are not found in the gold standard. Simple pseudocode for calculating this number is given in Figure 6.1(c). The upper bounds for both languages are given in the baseline tables.

Comparison With Previous Work

Similar to the baselines above, our upper bound calculations for WSJ10 match those reported in (Klein 2005) and our NEGRA results are nearly identical. While (Klein 2005)



(a) Treebank Tree (b) Binary Tree

1: $brackets_{gold} \Leftarrow 0$
2: $brackets_{extra} \Leftarrow 0$
3: **for each** *subtree* in $trees_{gold}$ **do**
4: $brackets_{gold}$++
5: **if** $|subtree.children| > 2$ **then**
6: $brackets_{extra} \stackrel{+}{\Leftarrow} |subtree.children| - 2$
7: **end if**
8: **end for**
9: $precision_{ubound} \Leftarrow \frac{brackets_{gold}}{brackets_{gold}+brackets_{extra}}$

(c) Pseudocode

Figure 6.1: Pseudocode for upper bound calculation

does not provide labeled upper bounds, (Haghighi & Klein 2006) does. Upper bound labeled recall is limited due to unary chains. Our labeled upper bound recall was measured slightly higher than previously reported, but as both numbers are above 0.90, we are unlikely to reach either level reported.

6.3 Uninformed EM

Also found in Tables 6.1 and 6.2 are two experiments labeled "EM." These systems represent prototype-free systems used both to replicate previous work, and to determine what contribution, if any, may be made by adding prototype constraints.

These uninformed runs use the full tagsets as given in Appendix C for their respective corpora and an addition of random noise, reported at their peak score. While these settings are standard, we wished to investigate how each setting might individually affect the system.

WSJ10 NOISE SETTINGS

	Labeled			Unlabeled		
	Prec.	Rec.	F_1	Prec.	Rec.	F_1
$t = 0$	16.26	20.64	18.19	38.92	49.42	43.55
$t = 0.01$	35.91	45.59	40.18	42.40	53.84	47.44
$t = 0.1$	36.54	46.40	40.89	43.33	55.02	48.48
$t = 1.0$	37.00	46.98	41.40	44.64	56.69	49.95
$t = 1000$	36.71	46.61	41.07	44.55	56.57	49.85

Table 6.3: Results of initializing the grammar with different amounts of noise. The value of t represents the threshold of noise added to each grammar rule as described in 4.3.1

6.4 EM Settings

Among the variables affecting the EM algorithm, we found that there were a number of settings that, while simple, could have a significant impact on induced results. In particular, we wished to find the desired level for random noise added to the grammar, the effects of changing the number of grammar symbols used, and what number of iterations produced optimal results.

6.4.1 Random Noise Experiments

When running the EM algorithm, it is common practice to add a small amount of random noise to the initial grammar in order to break the symmetry of a uniform distribution.

We were unable to find in the literature, however, precisely what levels of noise were helpful and whether too much noise would hurt more than it helped. We performed experiments to determine the effects of different amounts of randomization, following the method of adding random noise outlined in 4.3.1.

Analysis

As shown by the experiment labeled "$t = 0$" in Table 6.3, we found, as previously reported, that a uniform distribution performs terribly. Without some random noise to break the symmetry, the system remains stuck with poor parameter settings and yields a poor F_1

score of only 0.1819. As seen by other values for the noise threshold t in the table, noise levels above 0.01 seem to have little effect when averaged over runs, though noise levels over $t = 1000$ begin to decrease the scores.

6.4.2 Tagset Experiments

In addition to the variation possible due to methods of randomization, we tried varying the size of the tagset. Reducing the number of symbols in the tagset may be motivated for many reasons, two of the most important of which are performance and scalability. Lists of the symbols used and sizes of the tagsets used in this thesis can be found in Appendix C.

Performance

With respect to performance, for each nonterminal in the grammar, following the grammar definition given in A.2, all rules that can match $V_N \rightarrow VV$ will be generated. Since V is the set of both nonterminals and preterminals, the number of rules that are generated is equivalent to $|nonterminals \cup preterminals|^2$. Thus, decreasing the number of nonterminals in the grammar improves performance exponentially. Extra terminals give a performance hit as well, but rules with unused terminals are eliminated after the first EM iteration after not being seen in the training corpus. Nonterminals, however, are the classes we attempt to assign production rules to, so they are generally not removed unless hypotheses for some other class is so strong that all productions such an extra class might cover are entirely associated with other classes. As a result, the removal of nonterminals, if possible, is quite desirable.

Terminals in the tagset have different issues. While their presence may cause a decrease in performance, at least in early iterations, they intuitively provide more fine-grained evidence for inducing similarity between classes. For instance, the English Gerund VBG can behave in the same way as verbal nouns NN to form NPs, but can occur in contexts that Nouns don't. (**cf.** "Fast$_{JJ}$ running$_{NN}$ wins$_{VBZ}$ races$_{NNS}$" with "Running$_{VBG}$ quickly$_{RB}$ wins$_{VBZ}$ races$_{NNS}$") While the probability of *VBG RB* being an NP is likely to be high, if VBG is collapsed with NN, a bracketing of the terminals *NN RB VBZ NNS* is more likely

WSJ10 TAGSET REDUCTION

	Labeled			Unlabeled		
	Prec.	Rec.	F_1	Prec.	Rec.	F_1
NONTERMINALS	38.23	48.54	42.77	46.11	58.55	51.58
TERMINALS	36.37	46.20	40.70	46.00	58.43	51.47

Table 6.4: Effects of reducing the number of symbols contained in the tagset. NONTERMINALS shows the effect of reducing the number of nonterminals in the tagset, while TERMINALS shows the effect of reducing the number of terminals (POS Tags) in the tagset.

to group RB with a VP to the right (as in "He often wins races").

On the other hand, English participles (such as VBN) behave almost entirely as adjectives, and contextual clues they provides might not be damaged were it to be collapsed with JJ. To test the effects of both these cases, we ran experiments with tagsets that separately collapsed nonterminals and terminals to see the effects.

Scalability

As for scalability, our stated goal is to produce a system capable of analysis on multiple languages. While the WSJ10 tagset differentiates *Wh-* noun phrases, in languages where this does not happen, a tagset that collapses both to a general NP category would be more desirable. If such a tagset is possible without significant detriment to the produced parses, it would be helpful when seeking to expand the system.

Analysis

Table 6.4 shows the results of varying the number of nonterminals and terminals in the grammar. The full WSJ10 tagset contains 25 nonterminal and 35 terminal symbols, as given in Section C.1. The experiment labeled "NONTERMINALS" maintains all 35 terminal symbols, but reduces the number of nonterminals to the 8 given in Section C.2. The experiment labeled "TERMINALS" maintains the 25 nonterminals, but reduces the number of terminals to 13.

We were pleased to see that reducing the number of nonterminals actually improved

WSJ10 PROTOTYPES

	Labeled			Unlabeled		
	Prec.	Rec.	F_1	Prec.	Rec.	F_1
MANUAL	30.47	38.69	34.09	35.76	45.41	40.01
MANUAL+DIST	38.25	48.57	42.80	46.12	58.57	51.60
TREEBANK	40.35	51.24	45.15	46.63	59.21	52.17
TREEBANK+DIST	38.90	49.39	43.52	45.08	57.25	50.44

Table 6.5: Results of different methods of adding prototypes to the EM algorithm. MANUAL prototypes are a small set specified by a human. TREEBANK prototypes are automatically extracted from the WSJ10 gold standard using the methods described in 4.4.2. The +DIST property denotes the addition of additional prototypes using the distributional clustering method described in 4.4.3.

scores slightly from a labeled F_1 score of 0.4140 to 0.4277, rather than hurting them. We hypothesize that this result can be attributed to the EM algorithm's tendency to favor rarely-occurring rules for nonterminals, as such rules have a very low entropy, and help greatly towards maximizing the probability of the observed data. When there are a large number of nonterminal symbols used, this allows the algorithm to find many of these spurious classes, whereas reducing the available nonterminal symbols, constrains the algorithm to find fewer classes.

Reducing the number of terminals, on the other hand, produced a slight drop in scores, from a labeled F_1 score of 0.4140 to 0.4070. We believe this result to be expected per the performance discussion above, as more coarsely-grained yields will likely have problems forming classes similar to those in the gold standard as they contain less differentiating evidence.

6.4.3 Number of Iterations and Convergence

As noted in Chapter 3.6, it is common practice in grammar induction to set some threshold of change in log likelihood between EM iterations beneath which convergence is considered to be reached. While reaching this threshold often indicates subsequent iterations' performance will be asymptotic, it does not guarantee optimal performance due to the nature of the algorithm's objective function.

In order to determine at what point optimum performance may be reached, we ran a

number of experiments to 100 iterations and above and graphed their results, which can be found in Appendix E. We found that the prototype-influenced systems exhibited greatly different behavior than the uninformed algorithm, reaching a peak quickly near iteration 30, then declining with further iterations. The uninformed algorithm, however, took much longer to reach a similar peak, usually around 60 iterations. Furthermore, in the case of English, this optimal peak was reached after a plateau in iterations 0-30 iterations followed by a rapid climb in iterations 30-60.

Analysis

We are unsure what the cause of this behavior is in the uninformed algorithm, but it appears clear that while the prototype systems are quick to take to the given constraints, they are in danger of over-fitting with increasing iterations.

6.5 WSJ10 Prototype Experiments

Finally, we wished to see the effect of adding prototype constraints had on the system. We began by attempting to replicate a subset of the experiments[1] in (Haghighi & Klein 2006) performed on the WSJ10 corpus before attempting the IGT-based extraction on the NEGRA10 corpus. For these experiments on English, there are two variables to examine: the prototype source and the utility of distributional clustering to include additional prototypes.

Prototype Source

In examining prototype performance, we wished to not only see the results from a manually specified list of prototypes, but also the result of using our gold standard treebanks to extract prototypes, as a form of upper bound on prototype performance. This type of oracle result may produce one of the best obtainable results using prototypes. The results of these experiments are found in Table 6.5.

[1]Our experiments are most closely similar to the results reported as PROTO × NONE.

Prototype Method	Tagset	# Of Prototypes
Manual, No Extension	Reduced	23
Manual, Extended w/Clustering	Reduced	2753
Treebank Extracted, No Extension	Full	59
Treebank Extracted, Extended w/Clustering	Full	8102

Table 6.6: The prototypes used in the WSJ10 experiments were either manually specified or automatically extracted from a treebank as described in Section 4.4.2. Tagsets were limited to the nonterminals given in the given prototypes. Manual-specified prototypes used a reduced set of nonterminals, while the treebank prototypes were extracted for all nonterminals in the treebank.

WSJ10 UNMAPPED

	Labeled		
	Prec.	Rec.	F_1
MANUAL	27.48	34.89	30.75
MANUAL+DIST	31.29	39.73	35.00
TREEBANK	28.00	35.56	31.33
$t = 1.0$	0.82	1.04	0.92

Table 6.7: These results were produced by skipping the usual many-to-one mapping done in the standard metric. Rather than matching labels that find labels corresponding roughly to a correct class, these are the results of the labels found corresponding *exactly* to the desired labels. The names match those of experiments given in the preceding tables.

Number of Prototypes

Another piece of data we could not find for comparison were the number of prototypes used when distributional clustering was enabled. The numbers in Table 6.6 show the counts of unique prototypes found in our system. Note that, when manual prototypes are specified, they contain a reduced tagset consisting of only those nonterminals for which a prototype is given, and one catchall class "MISC".

Analysis

Comparing the scores for the prototype systems in Table 6.5 with those for the uninformed EM runs in Table 6.3 unfortunately shows the prototype modifications not performing as well as had been hoped. While the manually specified prototypes with distributional clustering enabled produced a labeled F_1 score of 0.4280, a significant improvement over

PREVIOUSLY REPORTED ON WSJ10

	Labeled			Unlabeled		
	Prec.	Rec.	F_1	Prec.	Rec.	F_1
PCFG × NONE	23.9	29.1	26.3	40.7	52.1	45.7
PROTO × NONE	51.8	62.9	56.8	59.6	76.2	66.9

Table 6.8: Numbers reported by (Haghighi & Klein 2006). These scores correlate most closely with our scores labeled "EM" in Table 6.1 and MANUAL+DIST in Table 6.5, respectively.

the 0.3409 achieved by the manual prototypes on their own, its margin over the uninformed NONTERMINALS system at 0.4277 is statistically insignificant.

While this is somewhat disappointing, remember that as discussed in Chapter 5, these labeled scores are greedily mapped to the labels that they match most frequently in the gold standard. When this mapping is disabled, as in Table 6.7, the benefit of the prototype system can be seen clearly. While the uninformed algorithm is clearly unable to produce proper labels, the constraints provided by the prototype systems force the system to adhere more closely to the desired labels.

Finally, another interesting result is the comparison of the TREEBANK system with the TREEBANK+DIST system. While prototype expansion with the MANUAL+DIST system yielded a significant 8-point improvement over the MANUAL system, the TREEBANK+DIST system actually performed worse using this prototype expansion.

We believe that this reduction in score is likely due to the fact that the prototypes found in the TREEBANK system are already quite high-quality and specific, having been extracted from an annotated source. Attempting to improve upon these prototypes, we believe, led to many more prototypes, but few as reliable as those initially extracted, resulting in a misclassification of many yields due to these lower-quality prototypes.

Comparison with Previous Work

The results obtained by our prototype system were quite different from numbers previously reported. Table 6.8 shows the numbers reported by (Haghighi & Klein 2006). The PCFG × NONE system correlates to our EM system in Table 6.1, while the PROTO × NONE

system matches our MANUAL+DIST system in Table 6.5.

While our uninformed EM system scored far higher than the results reported for a similar system, 0.414 to 0.263, our MANUAL+DIST system scored some 14 points behind Haghighi & Klein's system, 0.568 to our 0.428.

While Haghighi has noted that the uninformed system was likely underreported[2], we are unsure what causes the lower numbers in our MANUAL+DIST system. We believe the discrepancy lies in the details of the prototype modification code. Unfortunately, the Stanford NLP group maintains its code as an internal development project, and we were unable to reuse the same implementation used in (Haghighi & Klein 2006). Still, the usage of prototypes suggests a benefit in assigning labels to bracketings.

6.5.1 Unmapped Experiments

Though our primary metric for the experiments utilized many-to-one mapping, we wanted to ensure that prototype information was indeed properly constraining the EM algorithm to the labels we expected. While we are disappointed that our uninformed EM tests appeared to perform as well in finding syntactic classes as the prototype-informed tests, the results in Table 6.7 show how prototype information was at least partially successful in propagating correct labels to the syntactic categories.

6.6 NEGRA10 Prototype Experiments

While the prototype experiments performed on the WSJ10 corpus were performed primarily for comparison with the results previously reported in (Haghighi & Klein 2006), numbers for a German system were not given. Instead, our experiments on the NEGRA10 corpus were aimed at two goals: determining the effect of moving between tagsets as required by structural projection, and finally, the level of performance achievable using IGT-extracted prototypes.

---

[2](Haghighi 2008, personal communication)

NEGRA10 TREEBANK EXTRACTED PROTOTYPES

	Labeled			Unlabeled		
	Prec.	Rec.	F_1	Prec.	Rec.	F_1
PATH A	35.32	63.72	45.45	42.15	76.02	54.23
PATH B	36.24	65.37	46.63	42.92	77.42	55.23
PATH C	35.26	63.61	45.37	41.96	75.69	53.99

Table 6.9: Results of various methods of remapping NEGRA's German tagset to the English tagset used in IGT projection. Path letters refer to those illustrated in Figure 6.2.

6.6.1 Remapping Experiments

The results in Table 6.9 are all treebank-extracted prototypes. Similar to the tagset reduction experiments for English shown in Table 6.4, these experiments are concerned with the German tagset. Instead of focusing merely on the number of terminals in the tagset, however, this data is a result of both tagset reduction and possible English bias, as they are the product of mapping the NEGRA tagset into a tagset compatible with the WSJ tags produced by the IGT extraction process. Since the idea of moving from one language's tagset to another seems possibly destructive, we desired to see what effect such a remapping had.

As illustrated in Figure 6.2, there are multiple stages at which this tagset conversion could be done. **PATH A** in 6.2(a) is essentially the basic EM setting, with no tagset conversion done. The full NEGRA tagset is used for everything from the Prototypes and training data to the final parses as shown in 6.3(b).

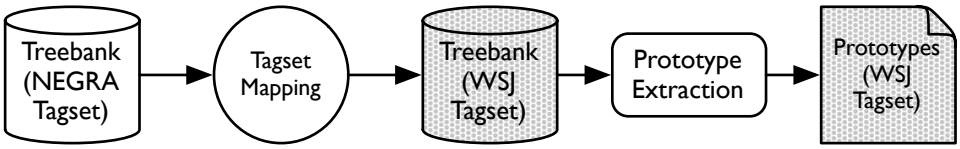
PATH B in 6.2(b) and **PATH C** in 6.2(c) are designed to determine the effects of tagset remapping necessary for using IGT-extracted prototypes. They both use prototypes extracted from the treebank, but PATH B extracts prototypes from a remapped treebank, while PATH C remaps prototypes initially extracted from the unmapped treebank.

Analysis

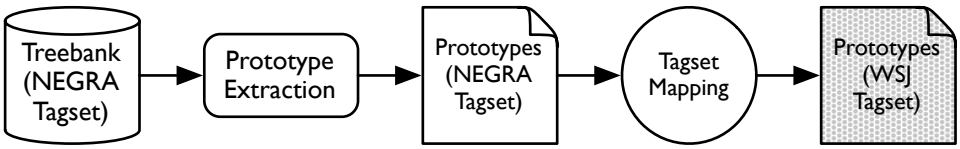
We found that PATH B Outperforms PATH C slightly. We believe that as PATH B remaps its treebank before extracting prototypes, this alters the counts of symbols in the treebank,

changing the prototypes that are output. Performing remapping on already-output prototypes as in PATH C is likely to cause discrepancies due to the late change in tagset, however.

(a) PATH A: In the simplest case, prototypes are extracted from the treebank and used as-is. This case requires the POS corpus to use NEGRA tags.

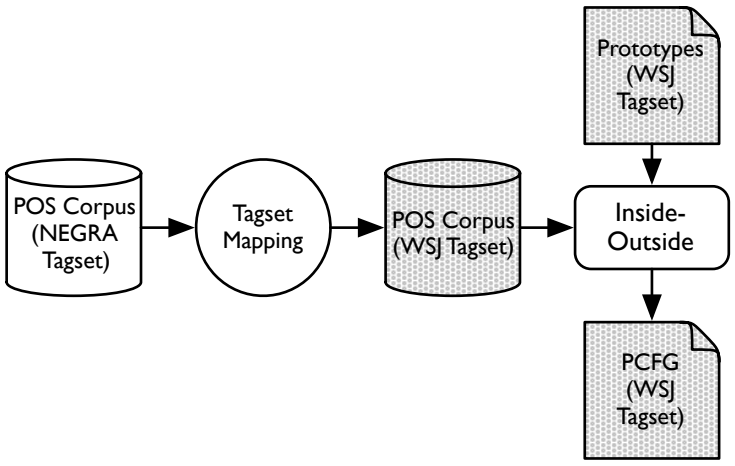


(b) PATH B: In this method, the treebank tagset is remapped *before* prototypes are extracted.

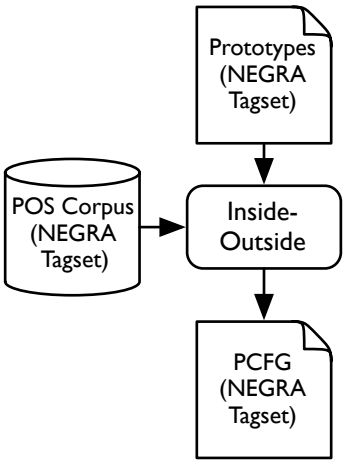


(c) PATH C: In this method, prototypes are extracted from the treebank, then the prototypes themselves are remapped.

Figure 6.2: Various "cheating" experiments that were performed on the NEGRA10 corpus. Approaches (b) and (c) will use a POS corpus that has similarly been remapped to the WSJ tagset, while (a) will use an unmapped POS corpus.



(a) When given prototypes using a WSJ tagset, the input POS corpus to the inside-outside algorithm must also have been remapped so that the prototype symbols will match those in the training data.



(b) When prototypes using the NEGRA tagset are used, no remapping is required to run inside-outside with prototype information. Furthermore, the resulting parses will use the NEGRA tagset.

Figure 6.3: Status of the tagset during training of the different systems. PATH A will use (b), while PATH B & C, and the IGT experiments, will use (a).

NEGRA10 - IGT EXTRACTED PROTOTYPES

	Labeled			Unlabeled		
	Prec.	Rec.	F_1	Prec.	Rec.	F_1
IGT	34.93	63.00	44.94	42.68	76.98	54.91
IGT+DIST	33.14	59.77	42.63	44.90	80.99	57.77

Table 6.10: Results of using IGT data from ODIN to extract prototype information.

6.6.2 IGT Experiments

Finally, we used the prototypes extracted from the projected IGT instances to inform the EM algorithm, both with and without distributional clustering. The results of these tests can be seen in Table 6.10.

Analysis

We found here, that while the addition of IGT prototypes improved performance over uninformed EM, extending the prototypes with distributional clustering actually hurt performance. We have two theories for why this may have happened, despite our +DIST system scoring better in English.

First, it is possible that this decrease in performance could be due to a compounding of errors introduced by the noise inherent in IGT. While we were able to extract some prototypes that showed correlation with correct tags, as seen in the *UNMAPPED* experiments, it is likely that these prototypes were of poorer quality than those extracted from the treebank or manually specified.

Since these IGT-extracted prototypes are given less weight than usual as described in 3.7.1, it is possible that a few incorrect prototypes could be effectively ignored by the EM algorithm, but when more weight is placed on them by means of greater coverage as in +DIST, these incorrect prototypes begin to penalize the system.

Second, as these IGT-extracted prototypes use the WSJ10 tagset, using these prototypes requires mapping the NEGRA-tagged POS corpus to a compatible WSJ-tagged corpus as illustrated in Figure 6.3(a). This step is required in order for the yields in the training

NEGRA10 UNMAPPED

	Labelled		
	Precision	Recall	F-Measure
IGT	20.96	37.80	26.97
IGT+DIST	16.76	30.23	21.56
EM	0.56	1.01	0.72

Table 6.11: Similar to Table 6.7, the results shown here show the results of the various experiments matching the gold standard's bracket labels *without* the use of many-to-one mapping.

data to correlate correctly with the specified prototypes. It is possible that performing distributional clustering on this remapped POS corpus leads to the addition of incorrect prototypes, resulting in poorer performance.

Ultimately, while the addition of these extra prototypes does not mirror the improvement shown by the English system, the IGT system without additional prototypes outperforms the uninformed EM algorithm shown in Table 6.2, showing a clear benefit.

6.6.3 Unmapped Experiments

Finally, the results in Table 6.11 show that while noisy, the presence of IGT-derived prototypes does cause the system to adhere more closely to the desired tagset than the uninformed system, which gets practically none of the labels correct when many-to-one mapping is not used.

6.7 Discussion

Though our results showed improvement when prototypes were added, the improvement was far below that reported by (Haghighi & Klein 2006). In a connected pipeline system such as ours, there are many potential sources for error; the two most likely are in the selection of prototypes, and the modification to the EM algorithm itself.

6.7.1 IGT

Cleaning

Perhaps one of the greatest limiting factors in using IGT instances as a source of prototypes is that the correct cleaning process throws out quite a large number of instances. Most of the time, these instances are cases where the number of tokens on the gloss line do not match the number of tokens on the language line, and the 1:1 nature of the gloss tags cannot be assured.

Furthermore, the current cleaning code relies heavily on regular expressions to help detect and clean data which does not appear to be IGT. Replacing this detection with something more flexible such as a classification algorithm would likely help.

Alignment & Projection

In addition to loss of data from cleaning, poor alignment is likely the cause of a good deal of the problems in IGT extraction. The existing code is intended primarily for lexicalized projection, and thus relies heavily on copies or morphs of the words in the translation line to be present on the gloss line.

When building a system such as ours that instead relies on POS tags, an aligner that takes into account more POS-related annotation on the gloss line may improve results. For instance, the tag `3sg` may be enough for that portion of gloss to be aligned with `VBZ` on the translation line. These improved alignments would result in fewer unaligned words, and thus ultimately better and more extracted prototypes.

Tagset Mapping

Although an attempt was made to map appropriate symbols from the NEGRA10 tagset to the WSJ10 tagset, a mistake in this mapping would alter the distributional patterns found in the POS corpus data. An alteration such as this could cause both the EM algorithm and the distributional clustering technique to be misled by constituents that did not behave like the intended class. A verb-like class mistakenly grouped as a noun could wrongly pressure the EM algorithm to include noun phrases in rules intended for verb phrases, for instance.

6.7.2 EM Algorithm

Though prototype extraction from IGT leaves room for improvement, perhaps the bigger issue was that of the EM results themselves. While our initial, fully unsupervised runs over the WSJ10 matched and even surpassed results reported in (Haghighi & Klein 2006), after adding prototypes our numbers fell far short of those reported for a similar system.

Such a deficit suggests an error with either the distributional clustering of prototypes or the constraint code. A bug in either of these systems would have been easy to mistake, given code complexity and the number of parameters involved in each. While we have attempted to provide results that offer a sanity-check on the system, we have been unable to rule out such logic errors, however.

Distributional Clustering

An error in the prototype expansion could have led to two scenarios; if the clusters were too strict and too few additional prototypes proposed, the algorithm could suffer due to data sparsity, since even with prototypical phrases the raw numbers of occurrences can be rather low. With too great a number of additional prototypes, non-constituent spans may be forced into being labelled together. This hypothesis seems to be likely, given the low precision numbers achieved by the system.

Constraint Modification

Finally, were there an error in the constraint code itself, when prototypes are specified the probabilities entered in the inside chart could have failed to skew the algorithm correctly. The fact that prototypes correctly tied the correct syntactic labels as seen in Table 6.7 makes this seem unlikely.

Chapter 7

CONCLUSION & FUTURE WORK

7.1 Summary of Results

After many experiments using different settings, our results are unfortunately mixed. Our experiments on the WSJ10 corpus designed to replicate previous work fell some 14 points below those reported by (Haghighi & Klein 2006), with our manual prototype system yielding a labeled F_1 measure of 0.428 to their reported 0.568, and only barely managed to beat the baseline of the uninformed EM system at 0.414.

On the other hand, the results from the NEGRA10 corpus are more promising, with the IGT Prototype experiments in Table 6.10 showing improvement over the uninformed EM in Table 6.2. Even so, the margin of improvement remains small, 0.4107 to 0.4494.

While our results have been mixed, they do show that IGT has the potential to inform grammar induction methods with some degree of success. To further this work, there are several avenues of research that remain to be investigated.

7.2 Expanding Language Coverage

Our IGT-extracted results for German showed potential, given that German is a language that has SOV word order, is strongly verb-second (V2), and has other syntactic dissimilarities from English. Despite that German and English belong to the same language family and are in many ways similar, differences such as these show hope that syntactic projection over IGT is sufficiently robust.

However, the next step we would have liked to pursue would be to use our system on Chinese, a language far more typologically dissimilar from English than German. Attempting to project structure from English to Chinese and extract prototypes would be a better test of our method's potential than German alone.

In addition, as our system strives to be cross-linguistically viable, other languages

have treebanks available, such as Arabic (Maamouri et al. 2004) and Hebrew (Itai, Winter, Altman & Nativ 2001), with even Hindi-Urdu on the way (*Workshop on Hindi-Urdu Treebank* 2009).

If IGT shows success with this broader selection of languages, it would greatly bolster the evidence of IGT as a helpful resource for yet untested languages.

7.3 Improved Projection & Extraction

While structural projection has shown to be a useful tool (Yarowsky et al. 2001) (Lewis & Xia 2008), projection from IGT has issues of noise to contend with. There are many ways the data itself could be improved, from using more machine-ready methods of encoding the data, such as XML (Hughes, Bird & Bow 2003) to better cleaning methods for existing data. Cleaner data means more successful alignment and more successful alignment means better extracted prototypes.

The projection algorithm itself may have room for improvement—perhaps even minimal human supervision such as providing typological information about the language could coerce more faithful word ordering that would result in better extracted prototypes.

Finally, though we have attempted to make our extraction algorithm noise-robust, input from a language expert such as basic word order could be used to give preference to certain extracted prototypes, or even automatically generate some.

7.4 Additional Induction Methods

Finally, while we have cited (Haghighi & Klein 2006) heavily, we have not implemented the CCM bracketer used in their highest-performing system. The results shown in their paper indicate that the inclusion of this system boosts F_1 measures an additional 6-9 points. With the addition of CCM to our system, we would hope to see even more promising results.

7.5 Conclusion

While there is obviously room for improvement, this first attempt at harnessing this novel source of supervision shows promise. That a meaningful parse tree of a data-poor language can be attempted using this approach has a great deal of potential. While for the time

being grammar induction remaining an intensely difficult challenge, it is exciting to see the potential of a new source of data in this arena.

Appendix A

GRAMMAR DEFINITION

A.1 Grammar Format

The format of the grammars used in experiments is very close to the standard definition:
$G = (V, \Sigma, R, S)$
Where V is the set of all nonterminals, Σ the set of terminals, disjoint with V, S designates the special nonterminal, the start state and R the set of rules relating from V to a series of symbols in the set $(V \cup \Sigma)$.

In an effort to make these PCFGs compatible with existing tools which tend to deal with lexicalized trees, ones that bottom out in words rather than part-of-speech tags, a category of **preterminal** V_P has been added. The set V_P is a set of nonterminals consisting of all the part-of-speech category labels, disjoint from the set of syntactic labels, which will be referred to as V_N. Preterminals can occur freely on the right side of a rule, but are limited to the left of rules which have a single right-hand child such as the following:

$$\text{NN} \rightarrow \textit{boy}$$

Rules such as these allow lexicalized grammars to also perform part-of-speech tagging. As is standard practice for grammar induction, however, it will be assumed that part-of-speech tagging has already been performed, as the state space of lexicalized grammars would be too complicated for unsupervised methods to perform well. As such, the set of terminals used are not lexical items such as *boy*, but rather part-of-speech labels that match their preterminal parents. This results in rules such as:

$\text{NN} \rightarrow \text{NN\_t}$ but not $\text{NN} \rightarrow \text{DT}$

This may seem an unnecessary step to take, as there will be only a single rule for each preterminal with 1.0 probability, but this is necessary for compatibility with off-the-shelf

tools expecting lexicalized grammars.

A.2 Grammar Restrictions

Using the symbols defined above, the format of the grammar is restricted to rules of the following form, (where V is the superset of V_N and V_P):

1. $S \rightarrow V_N$
2. $V_N \rightarrow V\ V$
3. $V_P \rightarrow t$

Appendix B

INSIDE-OUTSIDE ALGORITHM PSEUDO-CODE

Require: *sent*: input sentence

chart$_{inside}$: a chart used to represent inside probabilities for *sent*

chart$_{outside}$: a chart used to represent outside probabilities for *sent*

rules: A table where the current grammar's rules can be looked up by the symbols they contain.

expectedCounts: A table associating rules with their weights, as will be computed by inside-outside's implicit E-step.

```
1:  #seed the leaf nodes
2:  for word_i in sent do
3:     rules ⇐ (select rules where rule.rhs = word_i)
4:     for each rule do
5:        chart_inside[i][i + 1][rule.lhs] ⇐ rule.prob
6:     end for
7:  end for

8:  #Build the inside chart
9:  for start where sent.length-1 > start ≥ 0 do
10:    for end where start+1 < end ≤ sent.length do
11:       for split where start < split < end do
12:          cell_L ⇐ chart_inside[start][split]
13:          cell_R ⇐ chart_inside[split][end]
14:          for each symbol_L in cell_L.entries do
15:             for each symbol_R in cell_R.entries do
16:                prob_L ⇐ cell_L[symbol_L]
```

```
17:             prob_R ⇐ cell_R[symbol_R]
18:             rules ⇐ (select rules where rule.rhs[0] = symbol_L
                                        and rule.rhs[1] = symbol_R)
19:             for each rule do
20:               prob_inside ⇐ prob_L × prob_R × rule.prob
21:               chart_inside[start][end][rule.lhs] +⇐ prob_inside
22:             end for
23:           end for
24:         end for
25:       end for
26:     end for
27: end for

28: #Build the outside chart
29: start ⇐ 0
30: end ⇐ sent.length
31: for each rule_start where rule.lhs = symbol_start do
32:     chart_outside[start][end][rule_start.lhs] ⇐ 1.0    #seed the root cell of the outside chart.
33: end for

34: #Iterate over the cells, top down, filling in the outside chart.
35: loop
36:     cell_out ⇐ chart_outside[start][end]
37:     cell_in ⇐ chart_inside[start][end]
38:     #Consider possible left neighbors to this cell
39:     for left where 0 ≤ left < start do
40:       cell_parent ⇐ chart_outside[left][end]
41:       cell_L ⇐ chart_inside[left][start]
42:       for each symbol_parent in cell_parent.entries do
43:         for each symbol_L in cell_L.entries do
```

44: $rules \Leftarrow$ (**select** $rules$ where $rule.lhs = symbol_{parent}$ **and** $rule.rhs[0] = symbol_L$)
45: **for each** $rule_L$ **do**
46: $symbol_{this} = rule_L.rhs[1]$
47: $prob_{parent} \Leftarrow cell_{parent}[symbol_{parent}]$
48: $prob_L \Leftarrow cell_L[symbol_L]$
49: $prob_{out} \Leftarrow prob_{parent} \times prob_L \times rule_L.prob$
50: $prob_{in} \Leftarrow cell_{in}[symbol_{this}]$
51: $cell_{out}[symbol_{this}] \stackrel{+}{\Leftarrow} prob_{out}$
52: ***# This is where the rule counts are updated!***
53: $expectedCounts[rule] \stackrel{+}{\Leftarrow} prob_{in} \times prob_{out}$
54: **end for**
55: **end for**
56: **end for**
57: **end for**

58: *#Consider right siblings*
59: **for** $right$ where $end < right \leq sent.length$ **do**
60: $cell_{parent} \Leftarrow chart_{out}[start][right]$
61: $cell_R \Leftarrow chart_{in}[end][right]$
62: **for all** $symbol_{parent}$ in $cell_{parent}.entries$ **do**
63: **for each** $symbol_R$ in $cell_R.entries$ **do**
64: $rules \Leftarrow$ (**select** $rules$ where $rule.lhs = symbol_{parent}$ **and** $rule.rhs[1] = symbol_R$)
65: **for each** $rule_R$ in $rules$ **do**
66: $symbol_{this} \Leftarrow rule.rhs[0]$
67: $prob_{parent} \Leftarrow cell_{parent}[symbol_{parent}]$
68: $prob_R \Leftarrow cell_R[symbol_R]$
69: $prob_{out} \Leftarrow prob_{parent} \times prob_R \times rule_R.prob$
70: $cell_{out} \stackrel{+}{\Leftarrow} prob_{out}$
71: **end for**

```
72:         end for
73:       end for
74:     end for

75:     #Move to the next chart cell in the progression
76:     if start = sent.length − 1 then
77:       break
78:     else if start = end − 1 then
79:       start ⇐ 0
80:       end --
81:     else
82:       start++
83:       end++
84:     end if
85: end loop
```

Appendix C

TAGSETS

C.1 Full WSJ10 Tagset

Syntactic Tags (25)	
S	WHADJP
SBAR	WHNP
NP	SBARQ
VP	SQ
QP	UCP
SINV	
ADJP	
WHADVP	
CONJP	
PP	
PRT	
PRN	
RRC	
NX	
WHPP	
LST	
FRAG	
INTJ	
X	
ADVP	

POS Tags (35)	
PRP$	VBG
FW	VBN
WDT	VBP
LS	VBZ
WP	VBD
DT	VB
NN	JJR
NNS	JJ
NNPS	JJS
NNP	UH
RP	MD
POS	SYM
TO	IN
PRP	
RB	
WRB	
CC	
PDT	
RBS	
RBR	
CD	
EX	

C.2 Reduced WSJ10 Tagset

Syntactic Tags (8)
S
ADJP
ADVP
QP
NP
PP
VP
MISC

POS Tags (12)
NN
SCC
JJ
RB
IN
CD
CC
IND
DT
VB
QW
MISC

C.3 Full NEGRA10 Tagset

Syntactic Tags (22)
DL
CVZ
CAP
CVP
CPP
PP
NM
NP
AA
CH
CO
CNP
VP
AP
S
ISU
CS
VZ
MTA
MPN
AVP
CAVP

POS Tags (51)	
KON	VVIMP
PWAT	PWAV
NE	VMFIN
ADJA	ITJ
FM	VVIZU
APPO	PRF
NN	PDAT
APPR	PIAT
ADJD	VVINF
VAIMP	PTKVZ
PRELS	PIS
PROAV	VVFIN
APZR	ADV
PPOSAT	PPOSS
KOUS	PDS
KOUI	XY
PRELAT	PTKANT
PTKZU	CARD
PWS	VVPP
VMPP	ART
VMINF	VAFIN
PTKNEG	VAINF
APPRART	KOKOM
VAPP	PTKA
PIDAT	PPER
TRUNC	

Appendix D

NEGRA10 TO WSJ10 TAGSET CONVERSION TABLE

POS Tags

WSJ10 Tag	**NEGRA10 Tags**
PUNC	$, $*LRB*
CC	KON, KOUS, KOUI, KOKOM
VB	VVIMP, VVIZU, VVINF, VAMP
WDT	PWAT
WRB	PWAV
WP	PWS
NNP	NE
MD	VMFIN, VMINF
VBZ	VVFIN
JJ	ADJA
UH	ITJ
FW	FM
NN	NN
PRP	PDAT, PRF, PPER
IN	APPR, APZR, APPO, APPRART
DT	PIAT, PRELS, ART, PIS
FRAG	PTKVZ
ADV	ADV, ADJD, PROAV
PRP$	PPOSAT, PPOSS
TO	PTKZU
RP	PTKANT, PTKA, PTKNEG, TRUNC
PDT	PRELAT, PDS, PIDAT
SYM	XY
CD	CARD
AUX	VAFIN, VAINF, VAIMP
VBN	VAPP, VVPP, VMPP

Syntactic Tags

WSJ10 Tag	**NEGRA10 Tags**
PP	PP
QP	NM
NP	NP, MPN
FRAG	CH
VP	VP, VZ
ADJP	AP, AA, MTA
S	S, DL
PRT	ISU
ADVP	AVP

Appendix E

GRAPHS OF F_1 SCORES OVER ITERATIONS

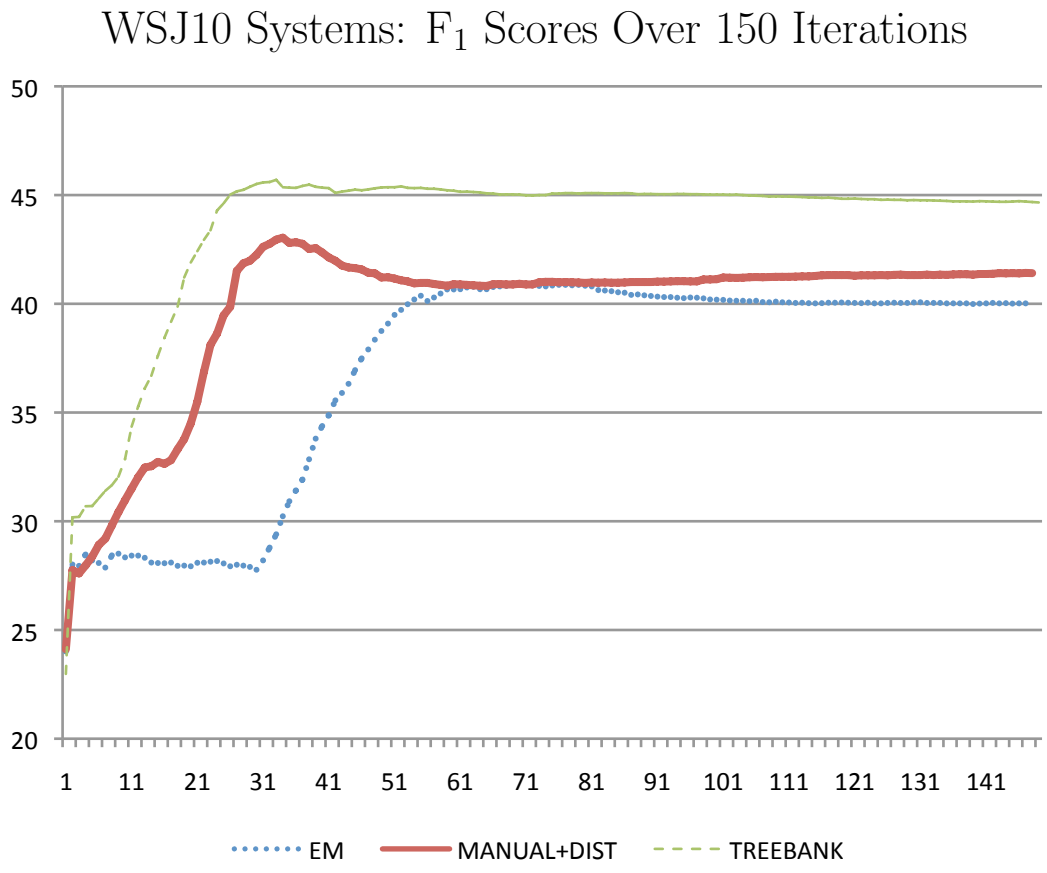


Figure E.1: Chart showing labeled F_1 scores for the WSJ10 uninformed EM, MANUAL+DIST, and TREEBANK experiments over 150 iterations.

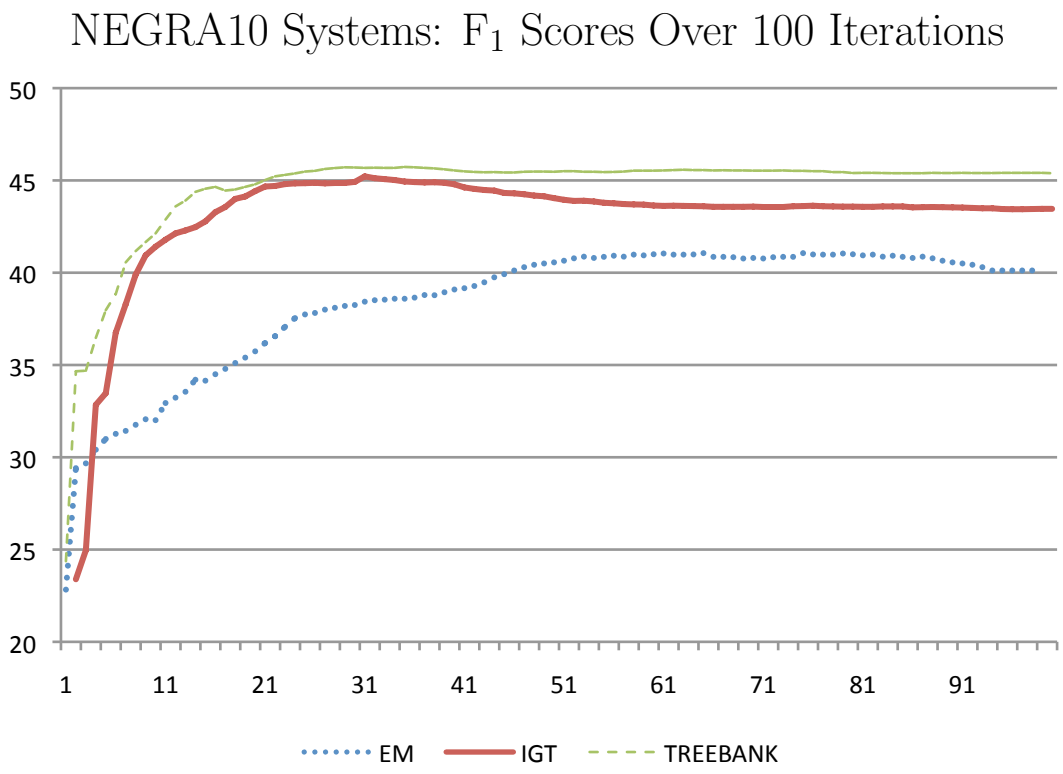


Figure E.2: Chart showing labeled F_1 scores for the NEGRA10 uninformed EM, IGT-extracted prototypes, and PATH B (treebank-extracted prototypes) experiments over 100 iterations.

BIBLIOGRAPHY

Baker, J. K. (1979), 'Trainable grammars for speech recognition', *The Journal of the Acoustical Society of America* **65**(S1), S132–S132.
URL: *http://link.aip.org/link/?JAS/65/S132/1*

Brill, E. (1993), Automatic grammar induction and parsing free text: A transformation-based approach, *in* 'Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics (ACL 1993)', pp. 259–265.

Carroll, G. & Charniak, E. (1992), Two experiments on learning probabilistic dependency grammars from corpora, *in* C. Weir, S. Abney, R. Grishman & R. Weischedel, eds, 'Working Notes of theWorkshop Statistically-Based NLP Techniques', AAAI Press, pp. 1–13.

Charniak, E. & Johnson, M. (2005), Coarse-to-fine n-best parsing and maxent discriminative reranking, *in* 'In ACL', pp. 173–180.

Clark, A. (2001), Unsupervised induction of stochastic context-free grammars using distributional clustering, *in* 'ConLL '01: Proceedings of the 2001 workshop on Computational Natural Language Learning', Association for Computational Linguistics, Morristown, NJ, USA, pp. 1–8.

Gordon, R. G., ed. (2005), *Ethnologue*, SIL International, Dallas.
URL: *http://linguistlist.org/pubs/books/get-book.cfm?BookID=14412*

Haghighi, A. (2008), 'personal communication'.

Haghighi, A. & Klein, D. (2006), Prototype-driven grammar induction, *in* 'Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics (COLING/ACL 2006', Association for Computational Linguistics, Sydney, Australia, pp. 881–888.

Hughes, B., Bird, S. & Bow, C. (2003), Encoding and presenting interlinear text using xml technologies, *in* 'University of Melbourne', pp. 105–113.

Ircs, N. X. (2002), 'Building a large-scale annotated chinese corpus'.
URL: *citeseer.ist.psu.edu/xue02building.html*

Itai, A., Winter, Y., Altman, A. & Nativ, N. (2001), 'Building a tree-bank of modern hebrew text. traitment automatique des langues'.

Johnson, M. (n.d.), 'Open-source software written by mark johnson'.
URL: *http://www.cog.brown.edu/ mj/Software.htm*

Klein, D. (2005), The unsupervised learning of natural language structure, PhD thesis, Stanford, Stanford, CA, USA. Adviser-Christopher D. Manning.

Klein, D. & Manning, C. D. (2002), A generative constituent-context model for improved grammar induction, *in* 'ACL', pp. 128–135.

Klein, D. & Manning, C. D. (2003), Accurate unlexicalized parsing, *in* 'ACL '03: Proceedings of the 41st Annual Meeting on Association for Computational Linguistics', Association for Computational Linguistics, Morristown, NJ, USA, pp. 423–430.

Lewis, W. D. (2006), Odin: A model for adapting and enriching legacy infrastructure, *in* 'In Proceedings of the e-Humanities Workshop'.

Lewis, W. D. & Xia, F. (2008), Automatically Identifying Computationally Relevant Typological Features, *in* 'Proceedings of The Third International Joint Conference on Natural Language Processing (IJCNLP)', Hyderabad.

Maamouri, M., Bies, A., Buckwalter, T. & Mekki, W. (2004), 'The penn arabic treebank: Building a large-scale annotated arabic corpus'.
URL: *citeseer.ist.psu.edu/maamouri04penn.html*

Marcus, M. P., Santorini, B. & Marcinkiewicz, M. A. (1994), 'Building a large annotated corpus of english: The penn treebank', *Computational Linguistics* **19**(2), 313–330.
URL: *citeseer.ist.psu.edu/marcus04building.html*

Maxwell, M. & Hughes, B. (2006), Frontiers in linguistic annotation for lower-density languages, *in* 'Proceedings of the Workshop on Frontiers in Linguistically Annotated Corpora 2006', Association for Computational Linguistics, Sydney, Australia, pp. 29–37.
URL: *http://www.aclweb.org/anthology/W/W06/W06-0605*

Pereira, F. & Schabes, Y. (1992), Inside-outside reestimation from partially bracketed corpora, *in* 'Proc. of the 30th Annual Meeting of the Association for Computational Linguistics (ACL 1992)', pp. 128–135.

Skut, W., Krenn, B., Brants, T. & Uszkoreit, H. (1997), An annotation scheme for free word order languages, *in* 'Proceedings of the Fifth Conference on Applied Natural Language Processing ANLP-97', Washington, DC.

Smith, N. A. & Eisner, J. (2006), Annealing structural bias in multilingual weighted grammar induction, *in* 'Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics

(ACL/COLING 2006)', Association for Computational Linguistics, Sydney, Australia, pp. 569–576.

Workshop on Hindi-Urdu Treebank (2009).
URL: *https://verbs.colorado.edu/hindi_wiki/index.php/Main_Page#Workshop_.28Hyderabad.2C_Jan.2709.29*

Xia, F. & Lewis, W. (2007), Multilingual structural projection across interlinear text, *in* 'Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference', Association for Computational Linguistics, Rochester, New York, pp. 452–459.
URL: *http://www.aclweb.org/anthology/N/N07/N07-1057*

Yarowsky, D., Ngai, G. & Wicentowski, R. (2001), Inducing multilingual text analysis tools via robust projection across aligned corpora, *in* 'HLT '01: Proceedings of the first international conference on Human language technology research', Association for Computational Linguistics, Morristown, NJ, USA, pp. 1–8.