EFFECTIVE STATISTICAL MODELS FOR SYNTACTIC AND SEMANTIC DISAMBIGUATION

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Kristina Nikolova Toutanova
September 2005

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Christopher D. Manning
(Principal Adviser)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Andrew Y. Ng

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Daniel Jurafsky

Approved for the University Committee on Graduate Studies.

Abstract

This thesis focuses on building effective statistical models for disambiguation of sophisticated syntactic and semantic natural language (NL) structures. We advance the state of the art in several domains by (*i*) choosing representations that encode domain knowledge more effectively and (*ii*) developing machine learning algorithms that deal with the specific properties of NL disambiguation tasks – sparsity of training data and large, structured spaces of hidden labels.

For the task of syntactic disambiguation, we propose a novel representation of parse trees that connects the words of the sentence with the hidden syntactic structure in a direct way. Experimental evaluation on parse selection for a Head Driven Phrase Structure Grammar shows the new representation achieves superior performance compared to previous models. For the task of disambiguating the semantic role structure of verbs, we build a more accurate model, which captures the knowledge that the semantic frame of a verb is a joint structure with strong dependencies between arguments. We achieve this using a Conditional Random Field without Markov independence assumptions on the sequence of semantic role labels.

To address the sparsity problem in machine learning for NL, we develop a method for incorporating many additional sources of information, using Markov chains in the space of words. The Markov chain framework makes it possible to combine multiple knowledge sources, to learn how much to trust each of them, and to chain inferences together. It achieves large gains in the task of disambiguating prepositional phrase attachments.

Acknowledgements

My thesis work would not have been possible without the help of my advisor, other collaborators, and fellow students. I am especially fortunate to have been advised by Chris Manning. Firstly, I am grateful to him for teaching me almost everything I know about doing research and being part of the academic community. Secondly, I deeply appreciate his constant support and advice on many levels. The work in this thesis was profoundly shaped by numerous insightful discussions with him.

I am also very happy to have been able to collaborate with Andrew Ng on random walk models for word dependency distributions. It has been a source of inspiration to interact with someone having such far-reaching research goals and being an endless source of ideas. He also gave me valuable advice on multiple occasions. Many thanks to Dan Jurafsky for initially bringing semantic role labeling to my attention as an interesting domain my research would fit in, contributing useful ideas, and helping with my dissertation on a short notice. Thanks also to the other members of my thesis defense committee – Trevor Hastie and Francine Chen. I am also grateful to Dan Flickinger and Stephan Oepen for numerous discussions on my work in HPSG parsing.

Being part of the NLP and the greater AI group at Stanford has been extremely stimulating and fun. I am very happy to have shared an office with Dan Klein and Roger Levy for several years, and with Bill McCartney for one year. Dan taught me, among other things, to always aim to win when entering a competition, and to understand things from first principles. Roger was an example of how to be an excellent researcher while maintaining a balanced life with many outside interests.

I will miss the heated discussions about research and the fun at NLP lunch. And thanks to Jenny for the Quasi-Newton numerical optimization code. Thanks to Aria Haghighi for collaborating on semantic role labeling models and for staying up very early in the morning to finish writing up papers.

I would also like to take the chance to express my gratitude to my undergraduate advisor Galia Angelova and my high-school math teacher Georgi Nikov. Although they did not contribute directly to the current effort, I wouldn't be writing this without them.

I am indebted in many ways to Penka Markova – most importantly, for being my friend for many years and for teaching me optimism and self-confidence, and additionally, for collaborating with me on the HPSG parsing work. I will miss the foosball games and green tea breaks with Rajat Raina and Mayur Naik. Thanks also to Jason Townsend and Haiyan Liu for being my good friends.

Thanks to Galen Andrew for making my last year at Stanford the happiest. Finally, many thanks to my parents Diana and Nikola, my sister Maria, and my nieces Iana and Diana, for their support, and for bringing me great peace and joy.

I gratefully acknowledge the financial support through the ROSIE project funded by Scottish Enterprise under the Stanford-Edinburgh link programme and through ARDA's Advanced Question Answering for Intelligence (AQUAINT) program.

Contents

List of Figures

Chapter 1

Introduction

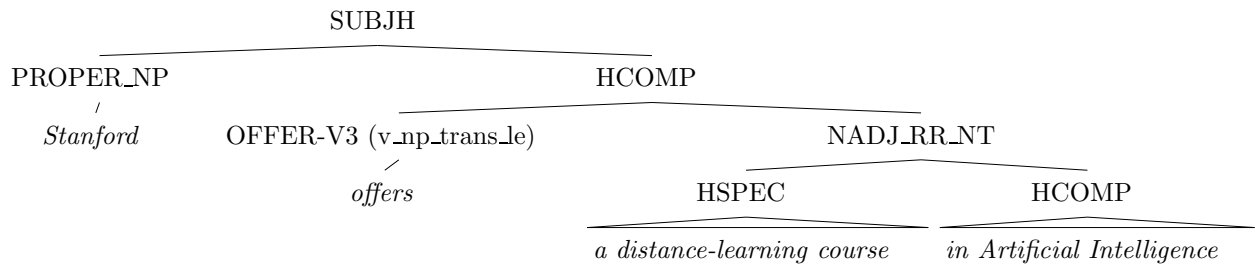
In recent years many natural language processing tasks have reached levels of wider applicability and demand. For example, in Information Extraction, instances of a specific relation – e.g., books and their authors, are automatically extracted from a collection of documents. The ability to collect such information is very desirable because of the huge document collections available in digital form. Another application which is gaining importance is answering questions from text. A multitude of information is accessible online, but current Information Retrieval technology is sometimes helpless in giving us the answers we seek.

Domain-specific heuristics have been developed to achieve acceptable levels of performance for information extraction in limited domains. However, to build broad coverage systems, we need to have ways of solving more general disambiguation problems of natural language syntax and semantics. For example, consider a system that aims to answer a broad range of questions about actions and their participants, such as:

Which universities offer distance-learning courses in Artificial Intelligence?

Evidently such a system needs broad-coverage semantic representation of English sentences.

One approach to coming up with semantic representations for sentences has been



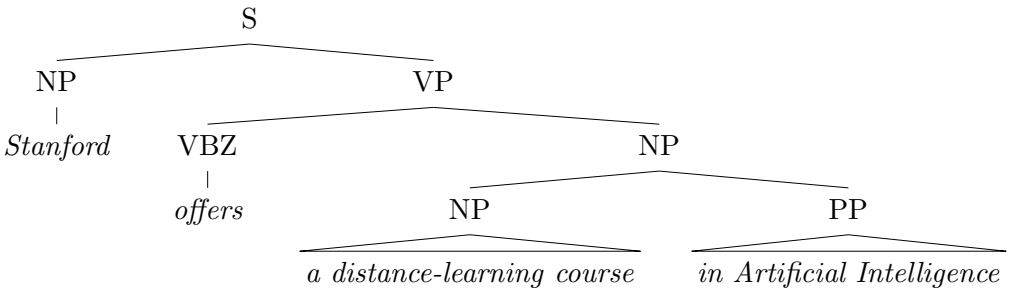
(a) HPSG derivation representation.

{ e2:
x4:named_rel[CONST_VALUE:Stanford]
_1:def_np_rel[BV x4:named_rel]
e2:_offer_v_rel[ARG1 x4:named_rel,ARG3 x13:_course_rel]
_2:_a_quant_rel[BV x13:course_rel]
e19:_distance_learning_rel[ARG x13:course_rel]
e24:_in_rel[ARG x13:course_rel ARG3 x22:intelligence_rel]
e26:_artificial_rel[ARG x22:intelligence_rel]
_4:prpstn_rel[SOA e2:_offer_v_rel]
}

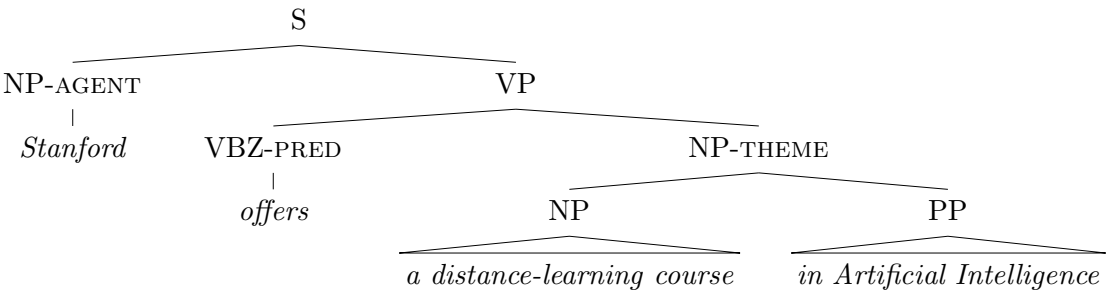
(b) HPSG semantic representation

Figure 1.1: An HPSG derivation and a schematic semantic representation corresponding to it for the sentence *Stanford offers a distance-learning course in Artificial Intelligence.*

to hand-construct a grammar, which can map each sentence to a set of possible structures. The structures contain both syntactic and semantic information, in enough detail to provide the basis for semantic inference for the natural language task at hand. Examples of this approach are precise grammars, such as Lexical Functional Grammar (Bresnan, 2001), Combinatory Categorial Grammar (Steedman, 1996), and Head-Driven Phrase Structure Grammar (HPSG) (Pollard and Sag, 1994). Figure 1.1 shows an HPSG analysis of the sentence "Stanford offers a distance-learning course in Artificial Intelligence", which if understood, would give a useful piece of information to the question answering system. The semantic representation which is specified as part of the HPSG analysis is shown in 1.1(b).



(a) Penn Treebank syntactic representation.



(b) Propbank semantic representation as an additional layer.

Figure 1.2: A syntactic Penn Treebank-style parse tree and Propbank-style semantic annotation for the sentence *Stanford offers a distance-learning course in Artificial Intelligence*.

Another approach has been to annotate a large collection of sentences with structural information which is thought to be useful for general NL tasks. In this way we do not have an explicit grammar but only a corpus grammar – examples of sentences with their corresponding structures. The most widely used structurally annotated corpora of this form is the Penn Treebank (Marcus et al., 1993). The parsed version consists of about 1 million words of newswire, which were initially annotated with primarily syntactic information. Subsequently, an additional layer of (shallow) semantics – the PropBank annotation (Palmer et al., 2005) – was added starting in 2002. Figures 1.2(a) and 1.2(b) show a Penn Treebank-style syntactic and semantic analysis of the same sentence.

For both approaches, we need to solve an enormous disambiguation problem –

to decide, for a given sentence, which of the many possible structures represents the meaning of the sentence correctly. The disambiguation problem is easier for hand-built precise grammars, because the grammar limits the number of possible analyses; however, even for them a sentence often has hundreds or thousands of possible analyses. Additionally, a problem with such grammars is that they are difficult to maintain and grow to cover broad enough domains.

The past 20 years of NLP research have shown that the best performing disambiguation systems have been based on learning from annotated data. In this work, we are concerned with learning statistical disambiguation models for syntax and semantics in both settings – for hand-built precise grammars, and for treebank grammars. In the first setting, given a sentence s and a set of possible analyses $T(s) = \{t_1(s), t_2(s), \ldots, t_m(s)\}$, and training data D annotated with sentences and their correct analyses, the task is to come up with a function $f(s, T(s)) \mapsto t_i(s)$ that decides which of the possible analyses is correct. The analyses can be very complex structures such as HPSG signs.

In the second setting, given a sentence s and a set of annotated data D, which specifies syntactic and semantic analyses for a collection of sentences $[s_i, t(s_i), sem(s_i, t(s_i))]$, the task is to come up with a function $f(s) \mapsto [t(s), sem(s, t(s))]$, which decides the correct syntactic and semantic analyses of s. For Penn Treebank annotations, there has already been a large amount of work on machine learning for syntactic parsing, i.e., learning functions $f_{syn}(s) \mapsto t(s)$, for example, (Magerman, 1995; Collins, 1997; Charniak, 2000). Therefore, to solve the task of finding function f, it suffices to learn a function $f_{sem}(s, t(s)) \mapsto sem(s, t(s))$. This is the approach we take in this work. We only learn a function that maps from syntactic to semantic analyses, and reuse existing state of the art models for syntactic parsing.

For both setting, the disambiguation task is very different from standard machine learning classification tasks. The hidden labels are large structured objects. Choosing a good feature representation of the target classes is crucial. These issues have been extensively explored in previous work on parsing.

However, for disambiguating new kinds of syntactic analyses, such as HPSG signs,

existing statistical models may be less appropriate. In addition, even though multiple schemes for decomposing trees into features have been proposed, the space of possible decompositions is far from fully explored. New advances in machine learning allowing novel representations of structured objects using kernels have been under-explored as well. One of the major contributions of this thesis is a representation of parse trees and a method for defining tree kernels, applied to HPSG disambiguation. We briefly sketch the ideas and results in §1.1, and present the models in depth in Chapter 2. Even though the representation was developed for HPSG signs, it is likely to bring gains to Penn Treebank parsing models as well, because it contains several novel aspects which capture general properties of syntax.

The problem of learning a mapping from syntactic to sematic representations, such as Penn Treebank syntax and PropBank semantics, has been extensively studied in the past 5 years. Similar work includes work on FrameNet (Baker et al., 1998), which has a related corpus and annotation scheme. Even if it turns out that it is advantageous to learn models performing simultaneous syntactic and semantic disambiguation, the problem of learning to map from syntactic to semantic representations, or more generally, from one structural representation to another, will still be very important. The reason is that as we come closer to the exact form of semantic annotations we need for particular NLP tasks, it will be advantageous to be able to annotate some small amount of data for the particular annotation type needed, and still be able to reuse the large amounts of data already annotated for previously defined representations. That is, we wouldn't need to annotate all previously existing corpora that researchers have used with the new annotation scheme. The problem of mapping from one complex representation to another is fascinating and broadly applicable in NLP. We sketch the main ideas and results in our work on syntax to semantics mapping in §1.2. Chapter 3 discusses this work in detail.

The problems of syntactic and semantic disambiguation are so hard because of the exponential number of possible structures and the idiosyncracies of individual words and phrases. Since every word has its own different semantics and often very specific syntax, the more reliable knowledge we can collect for individual words, the better our models will perform. The data sparsity problems in collecting word-specific statistics

are enormous. We propose a method of improving the estimation of word specific distributions through the use of multiple outside sources of knowledge that allow us to follow chains of inference and derive estimates for data-poor words. We sketch the main ideas and results in §1.3. Chapter 4 discusses this work in detail.

The three major chapters of the thesis are self-contained and can also be read independently if the reader is interested in only some of the topics.

1.1 A New Representation for Natural Language Parse Trees

1.1.1 Motivation

Our new representation for parse trees is motivated through multiple factors.

Increasing the connection of features with words and localizing broader structural context

Most feature representation of parse trees for learning statistical models are centered around context free rules of the form $A \rightarrow \alpha$. Most of the features active for a particular parse tree contain only non-terminal or hidden-structure symbols, and make no reference to the input sentence. This is counter-intuitive for disambiguation tasks, where the problem is to decide the correct structure t for a given input sentence s. Multiple annotation schemes where features are annotated with constituent head-words have resulted in more accurate models (Charniak, 1997; Collins, 1999; Charniak, 2000). However there are many instances of important non-head dependencies (Bod, 1998). Our proposed representation increases further the connection of the features with the input words in an intuitive fashion.

Previous work on statistical parsing has shown that structural, in addition to lexical annotation of the basic context-free rule features, is very useful to improve the performance of learned models. For example, using several ancestors (Charniak, 1997; Collins, 1997; Johnson, 1998) or other internal and external category splitting schemes (Klein and Manning, 2003) have increased performance dramatically. However, the

number of ancestors is still fixed and limited and some important longer-distance dependencies may be missed.

Developing representations suitable for HPSG analyses

As we can see from Figure 1.1(a), the derivation tree structure of an HPSG analysis, which shows the sequence of applied rule schema looks quite different from Penn Treebank parse trees. Most existing statistical models have been developed for the latter, with a few notable exceptions such as models for LFG (Johnson et al., 1999; Riezler et al., 2000), Combinatory Categorial Grammar (Hockenmaier, 2003b; Clark and Curran, 2004), Tree-Adjoining Grammar (Chiang, 2000), and previous work on HPSG parsing (Toutanova and Manning, 2002; Osborne and Baldbridge, 2004). Previously proposed models for HPSG parsing were mainly adaptations of Penn Treebank models.

An important characteristic of an HPSG grammar is that it is very lexicalized. The node labels, showing the rule schema, indicate the type of relationship of the head and non-head constituents which join at each node. Therefore it is worth exploring feature representations which are word-centered as well and depart to a larger extent from the context-free rule paradigm.

Permitting efficient computation with a very expressive feature set

Recent advances in machine learning and in particular the development of kernels for structured data, have made it possible to do efficient computation with exponential or even infinite feature spaces. Many kernels for strings have been developed and applied to text and other domains. Some kernels for graphs and trees have also been developed (see (Gärtner et al., 2002) for an overview). However, few of them are specifically motivated by natural language trees and more appropriate kernels could be defined. Our proposed linguistically motivated representation naturally extends to defining kernels which are centered around lexicalized broad structural fragments, and can build on previous work on string kernels.

1.1.2 Representation

The basic features we propose are paths from words to the top level of the tree, which we call "leaf projection paths". In HPSG such a path from a word will have one node for every modifier that the word takes (because the trees are binary).

As is always true for a grammar with non-crossing lexical dependencies, there is an initial segment of the projection path for which the leaf word is a syntactic head (called the *head path* from here on), and a final segment for which the word is not a syntactic head (called the *non-head path* from here on).

In traditional parsing models that have as features head word lexicalized local rules, some information present in the word projection paths can be recovered. Still, this is only the information in the head path part of the projection path. Our experiments show that the non-head part of the projection path is very helpful for disambiguation.

Using this representation of trees, we can apply string kernels to the leaf projection paths and combine those to obtain kernels on trees. The strings are sequences of nodes (with possible annotations), which are concatenated with either a word they correspond to, or its lexical type (part of speech). Using string kernels is even more advantageous when we have a multitude of features available to annotate each node. In HPSG analyses, each derivation tree node has an associated feature structure containing many features which could be useful for disambiguation. A kernel allows the fast computation of similarity measures based on the sub-features of nodes.

1.1.3 Summary of Results

Our experimental results are summarized below. We tested the models on the Redwoods corpus (Oepen et al., 2002) of HPSG-analysed sentences. The cited accuracy error reductions are for whole sentence accuracy – fraction of sentences for which the guessed analysis exactly matched the correct one.

- Models using non-head paths and head-paths had a 12.9% error reduction over models using only head-paths.

- Models using the leaf paths representation using tree kernels resulted in 13.8% error reduction over a model based on context free rules with an optimal level of parent annotation.
- The gain due to incorporating more sophisticated kernels as compared to simple n-gram kernels was 4.6%.

1.2 Mapping Syntactic to Semantic Structures

Semantic annotations for a large collection of sentences have only recently become available. Examples are the annotations produced by the FrameNet and PropBank projects. Even though the previously existing Penn Treebank provides syntactic information in the from of phrase structure annotations as exemplified in Figure 1.2(a), it is insufficient to capture information about actions and their participants. For example, the sentences "A distance-learning course in Artificial Intelligence is offered by Stanford" and "Stanford offers the general public a distance-learning course in Artificial Intelligence", are alternative ways of expressing the information that Stanford is the agent of offering with a theme "long-distance course in Artificial Intelligence". Yet Penn Treebank syntactic analyses alone are insufficient to allow the deduction of this information from the different realizations. The PropBank project defined an annotation scheme for representing shallow semantics of verbs, and labeled the parsed Wall Street Journal section of the Penn Treebank with an additional layer representing semantics of actions. An example annotation can be seen in Figure 1.2(b). If a parse tree node is labeled with a particular argument label – for example, THEME – this indicates that the phrase dominated by that node is a filler of the indicated semantic role for the verb marked PRED. For example, "a long-distance course in Artificial Intelligence" is a theme of "offers".

1.2.1 General Ideas

Previous work on mapping from syntactic to semantic annotations has identified many useful features (Gildea and Jurafsky, 2002; Pradhan et al., 2005a; Carreras and

Màrquez, 2005). Tremendous progress has been made in the performance achieved by such systems. The predominant approach has been to look at the mapping task as one of independent classification – assigning a semantic label to each node of a syntactic parse tree. We call such classifiers independent or *local*.

Linguistic intuition tells us that a core argument frame of a verb is a *joint* structure, with strong dependencies between arguments. For example, it is unlikely that multiple nodes will be labeled with the same semantic label, such as AGENT. In a typical parse tree, most of the parse tree nodes will not have a semantic label, but independent classification cannot explicitly model the count of argument labels. Also, it is never the case that nodes bearing argument labels are in a domination relationship. This is because of the natural constraint that the fillers of different semantic roles are different phrases. A local classifier cannot model this constraint and it must be externally enforced at test time. There are also strong statistical tendencies for the sequence of realized arguments. Even though previous work has modeled some correlations between the labels of parse tree nodes (Gildea and Jurafsky, 2002; Thompson et al., 2003; Punyakanok et al., 2005), many of the phenomena that it seems very important to capture are not modeled.

The main goal of our work is to look at the problem as one of learning a mapping from a syntactic to a joint semantic structure, which models correlations among multiple parts of the semantic structure. We ask the following questions:

1. What kind of global information can be usefully incorporated in models for mapping syntactic to shallow semantic structures?

2. How can such models be learned so as to overcome the computational complexity of learning and search?

We explore joint log-linear models incorporating novel global features connecting the syntactic tree to the complete semantic structure. Our proposed model is a Conditional Random Field (CRF) (Lafferty et al., 2001) with a very rich graphical structure. In contrast to previous work on CRFs for sequence classification tasks,

which has made strong markov independence assumptions on the label sequence, we build and do inference in a CRF without independence assumptions among labels.

To overcome computational complexity issues, we employ a re-ranking approach similar to (Collins, 2000). We also describe using dynamic programming to produce top N possible consistent semantic structures[1] according to the simpler local model.

The key properties of our proposed approach are: (*i*) no finite Markov horizon assumption for dependencies among node labels, (*ii*) features looking at the labels of multiple argument nodes and *internal features* of these nodes, and (*iii*) a statistical model capable of incorporating these long-distance dependencies and generalizing well.

1.2.2 Summary of Results

We compared our model to previous work on PropBank semantic role labeling. We mainly worked with the February 2004 preliminary version of the data. As it turns out, there are multiple issues in scoring the performance of such systems and the most reliable comparisons with previous work can be made using the CoNLL 2005 shared task evaluation (Carreras and Màrquez, 2005), where systems are trained using the same data and tested using the same evaluation measure. We also compare the results of our system to a *local* model, which classifies nodes in the parse tree independently and uses most of the features used in previous work. The only difference between the local and joint model is that the joint model incorporates dependencies among labels of multiple nodes. We obtain the following results:

- When the correct (gold-standard) syntactic parse trees are given, our joint model achieves an error reduction in F-Measure of 17% on ALL arguments and 36.8% on CORE arguments as compared to our local model.
- When automatic parse trees are used, our joint model achieves an error reduction in F-Measure of 8.3% on ALL arguments and 10.3% on CORE arguments as

[1]Consistent structures are ones that satisfy the constraint that argument labeled nodes are not in a domination relationship.

compared to our local model.

- On the CoNLL 2005 evaluation the error reductions of our joint compared to our local model are 18.4% on ALL arguments for gold-standard parse trees, and 7.8% on ALL arguments when using the best Charniak parse tree.

- The results of our submission to the CoNLL 2005 shared task were lower because of multiple issues discussed in Chapter 3, but our current results are 78.6%, 80.3%, and 68.8% on the development, Wall Street Journal test and Brown test set sets, respectively. These results on the test sets are about 1 F-measure point higher than the results of the winning system.[2]

1.3 Lexical Relations

A classic example of a kind of structural ambiguity that requires knowledge about lexical relations is prepositional phrase (PP) attachment. PP attachment decisions are also one of the major sources of ambiguity explosion in parsing. For example, in our sentence introduced above "Stanford offers a long-distance course in Artificial Intelligence", the prepositional phrase "in Artificial Intelligence" can either modify "course" (correct attachment), or it could modify "offers" as if it were specifying the location of offering (incorrect). Clearly, we need to have word-specific knowledge to decide the attachment – we need to know that "in Artificial Intelligence" is more likely to be a property of a course rather than a location or some other property of an offering event.

For many natural language tasks, the sparsity problems are severe as exemplified by work on language modeling and syntactic parsing. Multiple smoothing techniques have been developed and applied for language modeling ((Chen and Goodman, 1998) contains on overview). One of the insights for attacking the sparsity problem has been that it must be possible to use statistics about similar or related words, when information about a particular word is non-existent. There are multiple ways of

[2]It is possible that other teams have also improved their numbers in the meantime.

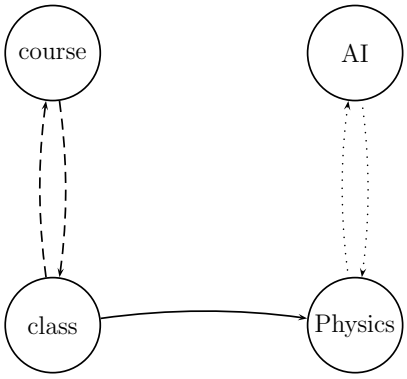


Figure 1.3: An example word space with similarities for the relation $in_(noun_1, noun_2)$.

defining similarities among words. One source is knowledge about word morphology, which tells us that words sharing the same lemma or stem are similar. Another source is manually constructed thesauri or word taxonomies, such as WordNet (Miller, 1990). These contain some very general knowledge about relationships among words, but do not have complete coverage. Additional sources could include statistics based on separate corpora, perhaps built for different tasks, but which could be the basis for useful sources of similarity measures.

Previous work has applied similarity measures in isolation and evaluated their contribution to particular tasks. Stemming and WordNet, for example, have been found helpful for some tasks and harmful for others.

1.3.1 General Ideas

We go a step further and propose a framework in which multiple similarity measures can be incorporated, and inferences based on them can be chained in multi-step chains of inference. Figure 1.3 illustrates an example chain of inference we may follow. Suppose that we want to estimate the probability that *AI* modifies *course* via the preposition *in*. In other words, we want to estimate the probability that

a word dependency relationship of the form $in(course, AI)$ occurs. Suppose also that we have access to WordNet, which tells us that *course* is a synonym to *class*. We have also observed the word dependency relationship $in(class, Physics)$ in our annotated corpus. Additionally, we have seen *Physics* co-occurring with AI in some type of documents on the web, which gives us another similarity measure. If we take these three inference steps starting from *course*, we will be able to conclude that $in(course, AI)$ is likely.

We propose to achieve such inferences via constructing a Markov chain in the state space of words. The target word dependency distribution is estimated as the limiting distribution of the Markov chain. The words are connected via different similarity measures. We automatically learn weights for the different similarity measures, in accordance with their usefulness for the task at hand.

The example in Figure 1.3 can be seen as a small state space of a Markov chain with only 3 types of links between words – synonym links from WordNet, co-occurrence links, and observed frequency links.

1.3.2 Summary of Results

Following most of the literature on PP attachment, starting with the work of Hindle and Rooth (1993), we focus on the most common configuration that leads to ambiguities: V NP PP. Here, we are given a verb phrase with a following noun phrase and a prepositional phrase. The goal is to determine if the PP should be attached to the verb or to the object noun phrase. For example, in the sentence "Stanford offers a long-distance course in Artificial Intelligence", this configuration occurs as:

$$[\textit{offers}]_V\ [\textit{a long-distance course}]_{NP}\ [\textit{in Artificial Intelligence}]_{PP}.$$

We evaluate our model on the standard IBM dataset (Ratnaparkhi et al., 1994) used in previous work on PP attachment. We incorporate several knowledge sources, in addition to the standard training set of annotated attachments.

Our baseline is a well-smoothed generative model for the word dependency relations involved in the ambiguity, trained using only the standard training and development sets. It is already significantly superior to previously proposed machine learning models, trained using only this information. The accuracy of our baseline model is 85.89%, and the best previous comparable result is 84.8% (Vanschoenwinkel and Manderick, 2003).

Incorporating additional knowledge sources in our Markov chain model resulted in significant gains in accuracy. The sources used were morphology, WordNet, and similarities extracted from an additional set of noisy prepositional phrase attachment examples. The additional set was constructed using a 30 million word text collection, parsed by the parser of Charniak (2000), and known as the BLIPP corpus. We extracted examples in the same form as the IBM training set from the parse trees of the BLIPP corpus. We refer to this set of examples as BLIPP-PP. In summary, our results were:

- Adding noun and verb morphology, and WordNet synonymy links for nouns and verbs improved the accuracy from 85.89% to 86.53%.
- Adding verb morphology, several types of distributional similarity links from BLIPP-PP, and empirical frequency links from BLIPP-PP improved the accuracy from 85.89% to 87.54%.

The upper bound in accuracy, estimated using human annotator agreement is 88.2% and the best reported result on this dataset is 88.1% (Stetina and Nagao, 1997). Our results are very close to these figures; additionally, differences of this size (about .6) were found not to be statistically significant for our models, as detailed in Chapter 4.

Chapter 2

Leaf Path Kernels for Natural Language Parse Trees

In this chapter we are concerned with building statistical models for parse disambiguation – choosing a correct analysis out of the possible analyses for a sentence. Many machine learning algorithms for classification and ranking require data to be represented as real-valued vectors of fixed dimensionality. Natural language parse trees are not readily representable in this form, and the choice of representation is extremely important for the success of machine learning algorithms. Here we concentrate on coming up with a good representation. In particular, we study the problem in the context of disambiguation for sentence analyses produced by a Head-driven Phrase Structure Grammar (HPSG). HPSG is a modern constraint-based lexicalist (or "unification") grammar formalism.[1] We evaluate our models on the Redwoods corpus of HPSG analyses (Oepen et al., 2002). Parts of this work were done in collaboration with Penka Markova and published in (Toutanova et al., 2004b). Previous work leading to the present models was described in (Toutanova and Manning, 2002; Toutanova et al., 2003; Toutanova et al., 2005b).

The main goals of our work are:

- Come up with a feature representation for syntactic analyses, which is well

[1] For an introduction to HPSG, see (Pollard and Sag, 1994).

suited to HPSG grammars.

- Improve the accuracy of models by incorporating more expressive and discriminative features.
- Incorporate large and even infinite-dimensional feature spaces to model properties of syntactic analyses, through the use of kernels.

We start with a brief introduction of preliminaries on statistical parsing and kernels.

2.1 Preliminaries

From a machine learning point of view, the parse selection problem can be formulated as follows:

The training set is given as m training examples of the form $(s^i, t^i_0, \ldots, t^i_{p_i-1})$. The number of possible parses for sentence s^i is p_i. For each sentence s^i, we also know the correct analysis; we will assume, without loss of generality, that the correct analysis for s^i is t^i_0. We are also given a feature representation $\Phi(t^i_j)$ for the syntactic analyses t^i_j. Most often, the feature representations are vectors in a finite-dimensional Euclidean space R^n.

The goal is to learn a classifier that can select the correct analyses for unseen sentences, that is, learn a function $f(s, t_0, \ldots, t_{p-1}) \mapsto \{t_0, \ldots, t_{p-1}\}$. The problem is usually solved by learning a discriminant function $g(\Phi(t_j)) \mapsto R$ and the analysis that maximizes that function for a given sentence is chosen as the guessed analysis:

$$f(s, t_0, \ldots, t_{p-1}) = \arg\max_j g(\Phi(t_j))$$

This problem differs from standard classification problems, because in classification a finite set of atomic classes is usually specified and these classes are possible for all inputs. In contrast, in parsing the possible analyses are completely disjoint across different sentences.

Both *generative* and *discriminative* models have been proposed for parse selection. In generative models, the joint distribution $P(s, t_j)$ is modelled and the discriminant function is $g = P(s, t_j)$. In discriminative models, the function g either models the conditional probability $P(t_j|s)$ or directly tries to maximize the accuracy of the resulting classifier f.

One class of models, widely used in parsing and other applications, is the class of linear models. In these models, the discriminant function is linear in the features., i.e., it is specified by a vector W, and the function $g(t_j) = \langle W, \Phi(t_j) \rangle$, where $\langle X, Y \rangle$ is the inner product of two vectors. The space of linear models is very general because by expanding the feature representation Φ we can obtain a very expressive hypothesis space. An excellent treatment of linear models for parsing can be found in (Collins, 2001). There are multiple machine learning methods for linear models, differing in the way they select the parameter vector W given a training set of disambiguated instances. Logistic regression, Probabilistic Context Free Grammars, Support Vector Machines, Perceptron, and Boosting are all examples of linear models.

Support Vector Machines (SVMs) are a type of linear models, which have been demonstrated to be highly competitive across many domains. SVMs also have the advantage of allowing efficient computation with very large feature maps Φ through the use of kernels (similarity measures), which we define in §2.1.2. The following subsection gives more details on SVM learning.

2.1.1 Support Vector Machines

The training criterion in SVMs (Vapnik, 1998) selects a linear hyperplane, which roughly maximizes the difference in scores of the correct and incorrect analyses. In particular, it maximizes the following quantity, called the margin:

$$\hat{\gamma} = \min_{i,j>0} \frac{\langle W, \Phi(t_0^i) \rangle - \langle W, \Phi(t_j^i) \rangle}{||W||^2}$$

We introduce the formulation of SVMs for parsing of (Collins, 2001), which is very similar to the multi-class SVM formulations of (Weston and Watkins, 1998; Crammer

and Singer, 2001). The optimization problem to be solved is as follows:

$$\min \frac{1}{2}\langle W, W\rangle + C\sum \xi_{i,j}$$

$$\forall i \forall j > 0 : \langle W, (\Phi(t_0^i) - \Phi(t_j^i))\rangle \;\geq\; 1 - \xi_{i,j}$$
$$\forall i \forall j > 0 : \xi_{i,j} \geq 0$$

The $\xi_{i,j}$ are slack variables used to handle the case where a linear hyperplane that classifies the training set correctly does not exit. The same formulation has been used for natural language parsing in (Collins, 2001; Shen and Joshi, 2003; Taskar et al., 2004).

SVMs belong to the large class of machine learning algorithms which only need inner products between examples and thus can be solved without explicitly specifying the feature vectors $\Phi(t_j^i)$. The optimization problem can be solved in the dual formulation and it requires only the inner products $\langle \Phi(t_{j_1}^{i_1}), \Phi(t_{j_2}^{i_2})\rangle$. Equivalently, it requires the specification of a kernel. The next subsection gives background on kernels.

2.1.2 Kernels

Let $\mathcal{X}$ be a set of objects (syntactic analyses in our case). A Kernel K is a function $\mathcal{X} \times \mathcal{X} \mapsto R$, with the following properties:

1. K is symmetric, i.e., $\forall x, y \in \mathcal{X}$, $K(x, y) = K(y, x)$.

2. K is positive definite, in the sense that for any $N \geq 1$ and any $x_1, \ldots, x_N \in \mathcal{X}$, the matrix $K_{ij} = K(x_i, x_j)$ is positive definite, i.e., $\sum_{ij} c_i c_j K_{ij} \geq 0$ for all $c_1, \ldots, c_N \in R$.

Note that if there exists a mapping $\Phi : \mathcal{X} \mapsto R^d$, $\Phi(x) = \{\phi_i(x)\}_{i=1,\ldots,d}$, such that K is the inner product in R^n, i.e., for every x and y, $K(x, y) = \langle \Phi(x), \Phi(y)\rangle$, then it is easily verifiable that K is a kernel. The same holds if K is the inner product for

a mapping $\Phi : \mathcal{X} \mapsto l_2$ instead, where l_2 is the Hilbert space of all square-summable sequences, i.e. $\sum_{i=1}^{\infty} \phi_i(x)^2 < \infty$.

Conversely, for a broad range of kernels it can be shown that such a mapping exists (Haussler, 1999). In particular, this is always true for countable sets $\mathcal{X}$, which is the case for syntactic analyses.

Therefore, if we come up with such a mapping Φ for a function K, we can be sure it is a kernel. We will mostly define kernels in this way. Additionally, the class of kernels has nice closure properties which allow us to construct new kernels from existing ones, using addition, product, etc. (Berg et al., 1984).

In this chapter we will use the construction of kernels using a convolution, which is useful for structured objects. This method was introduced in (Haussler, 1999). We will describe the construction, following Haussler (1999), without proof.

We only state the construction for countable sets, because this is sufficient for our purposes. Suppose $x \in \mathcal{X}$ is a composite structure and one decomposition into its parts is $\vec{x} = x_1, \ldots, x_D$, where each part $x_d \in X_d$, for $1 \leq d \leq D$. Each of the sets X_d is countable, as is $\mathcal{X}$. Let R represent the relation on $X_1 \times \cdots X_D \times \mathcal{X}$ defined as follows: $R(x_1, \ldots, x_D, x)$ is true iff $x_1, \ldots, x_D$ is a decomposition of x into parts. Let $R^{-1}(x) \stackrel{def}{=} \{\vec{x} : R(\vec{x}, x)\}$. The relation R is finite if $R^{-1}(x)$ is finite for all $x \in \mathcal{X}$.

To give a concrete example, suppose that $\mathcal{X}$ is the set of pairs of natural numbers $\mathcal{X} = N \times N$. Then $R(i, j, x)$ is true iff x is the pair of natural numbers $[i, j]$. We will later define a relation that decomposes trees into their parts.

Suppose we have kernels K_1, ..., K_D, defined on $X_1 \times X_1$,..., $X_D \times X_D$, respectively. We define the similarity $K(x, y)$ as the following generalized convolution:

$$K(x, y) \stackrel{def}{=} \sum_{\vec{x} \in R^{-1}(x), \vec{y} \in R^{-1}(y)} \prod_{d=1}^{D} K_d(x_d, y_d)$$

K is defined on $S \times S$, where S is the subset of X for which R^{-1} is non-empty. An R-*convolution* of K_1,..., K_D, denoted by $K_1 \star \cdots \star K_D$ is defined on $\mathcal{X} \times \mathcal{X}$ as the zero extension of $K(x, y)$ to $\mathcal{X} \times \mathcal{X}$. The zero extension of $K(x, y)$ is the same as

$K(x, y)$ on $S \times S$, and is defined to be 0 whenever x or y is not in S.

The version of the theorem for convolution kernels we will use is as follows: If $K_1, \ldots K_D$ are kernels on the countable sets $X_1 \times X_1, \ldots, X_D \times X_D$, respectively, and R is a finite relation on $X_1 \times \cdots X_D \times \mathcal{X}$, where $\mathcal{X}$ is countable, then $K_1 \star \cdots \star K_D$ is a kernel on $\mathcal{X} \times \mathcal{X}$.

2.2 The Redwoods Corpus

The dataset of this work is the Redwoods corpus (Oepen et al., 2002). The grammar formalism underlying the Redwoods corpus is Head Driven Phrase Structure Grammar, which is a kind of lexicalist unification-based grammar. In unification-based grammar, the lexical entries for words and the rules are represented as feature structures, or attribute-value matrices, which form a flexible mechanism for succinctly expressing linguistic constraints.

Figure 2.1 shows a simple example of a grammar rule and two lexical entries represented in a unification-based grammar. We can see that agreement is achieved very naturally without having to specify multiple context free rules for each combination of feature values.

The HPSG grammar used to parse the sentences in the Redwoods corpus is the LinGO ERG (English Resource Grammar). Information in HPSG is represented by a *sign*, a typed feature structure which represents phonological, syntactic, and semantic information about a word or phrase. This information is built up for a sentence compositionally from the signs of sentence parts. We have not used the full HPSG sign in our current models, but rather a number of simpler projections of the sign and how it was composed, which are available in the Redwoods corpus. Most similar to Penn Treebank parse trees are phrase structure trees projected from the sign (Figure 2.2(b)), but in this work we have concentrated on use of derivation trees (Figure 2.2(a)), which record the combining rule schemas of the HPSG grammar which were used to license the sign by combining initial lexical types.[2] The internal nodes

[2]This derivation tree is also the fundamental data stored in the Redwoods treebank, since the

$$\text{S} \quad \Rightarrow \quad \text{NP} \quad \text{VP}$$

$$\begin{bmatrix} \text{CAT} & \textit{noun} \\ \text{NUM} & x \\ \text{PERS} & y \end{bmatrix} \quad \begin{bmatrix} \text{CAT} & \textit{verb} \\ \text{NUM} & x \\ \text{PERS} & y \end{bmatrix}$$

$$\text{am} \quad \begin{bmatrix} \text{CAT} & \textit{verb} \\ \text{NUM} & \text{sg} \\ \text{PERS} & 1 \end{bmatrix}$$

$$\text{I} \quad \begin{bmatrix} \text{CAT} & \textit{noun} \\ \text{NUM} & \text{sg} \\ \text{PERS} & 1 \end{bmatrix}$$

Figure 2.1: A grammar rule and lexical entries in a toy unification-based grammar.

represent, for example, head-complement, head-specifier, and head-adjunct schemas, which were used to license larger signs out of component parts. These derivation trees hence provide significantly different information from conventional phrase structure trees, but have proven to be quite effective for disambiguation. These representations are more fine-grained than those familiar from the Penn Treebank: for example, rather than 45 part-of-speech tags and 27 phrasal node labels, we have about 8,000 lexical item identifiers, and 70 derivational schemas. The lexical types are more similar to those employed in Lexicalized Tree-Adjoining Grammar work (Srinivas and Joshi, 1999), encoding information such as verbal subcategorization.

Additionally, the (implicit) Penn Treebank grammar and the LinGO ERG differ in that the Penn Treebank often uses quite flat grammatical analyses while the ERG is maximally binary, with extensive use of unary schemas for implementing morphology and type-changing operations. Much common wisdom that has been acquired for building probabilistic models over Penn Treebank parse trees is implicitly conditioned on the fact that the flat representations of the Penn Treebank trees mean that most

full sign can be reconstructed from it by reference to the grammar.

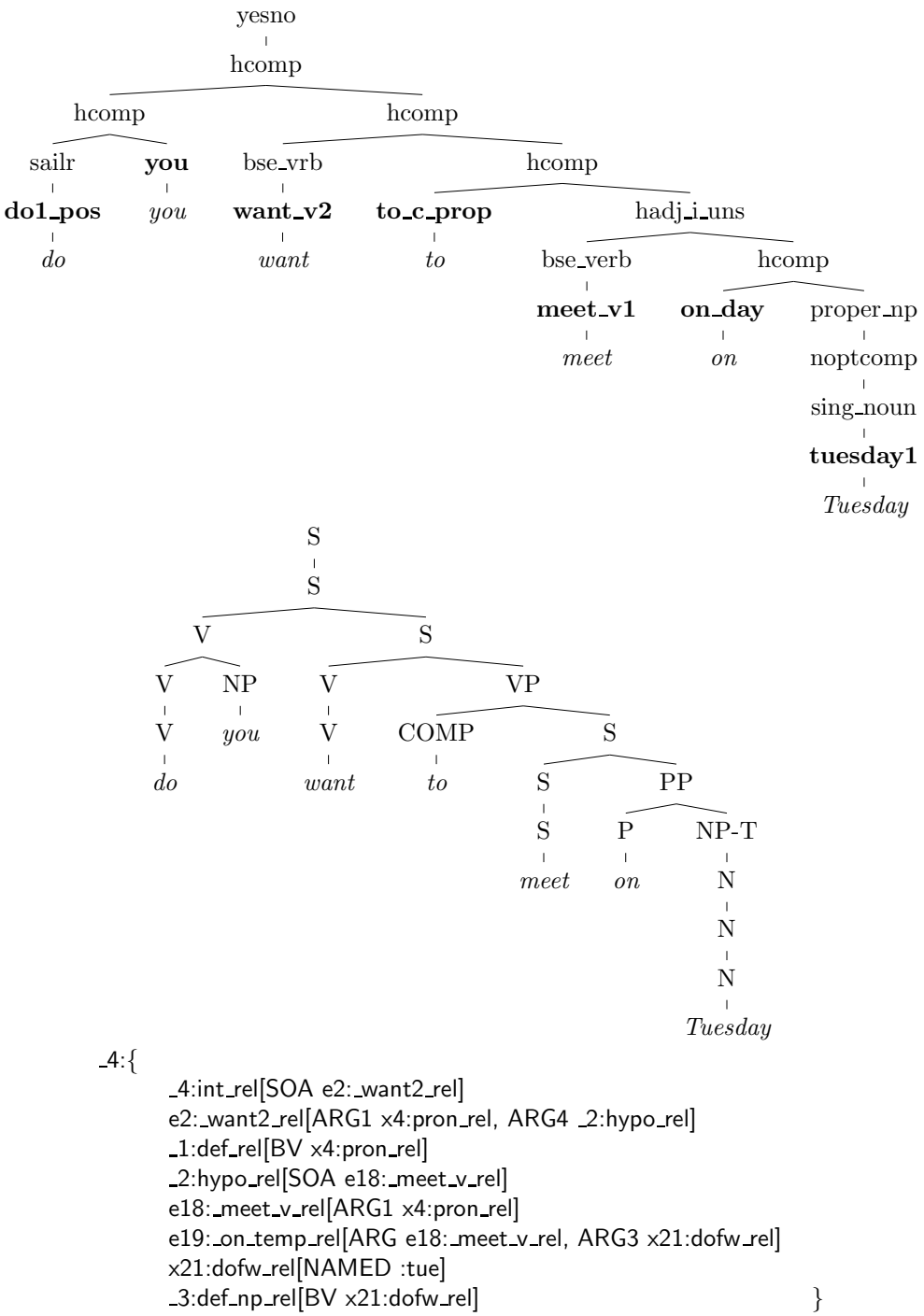


Figure 2.2: Native and derived Redwoods representations for the sentence *Do you want to meet on Tuesday?* — (a) derivation tree using unique rule and lexical item (in bold) identifiers of the source grammar (top), (b) phrase structure tree labelled with user-defined, parameterizable category abbreviations (center), and (c) elementary dependency graph extracted from the MRS meaning representation (bottom).

sentences	length	struct ambiguity
3829	7.8	10.8

Figure 2.3: Annotated ambiguous sentences used in experiments: The columns are, from left to right, the total number of sentences, average length, and structural ambiguity.

important dependencies are represented jointly in a local tree. Thus lessons learned there may not be applicable to our problem (see Collins (1999) for a careful discussion of this issue).

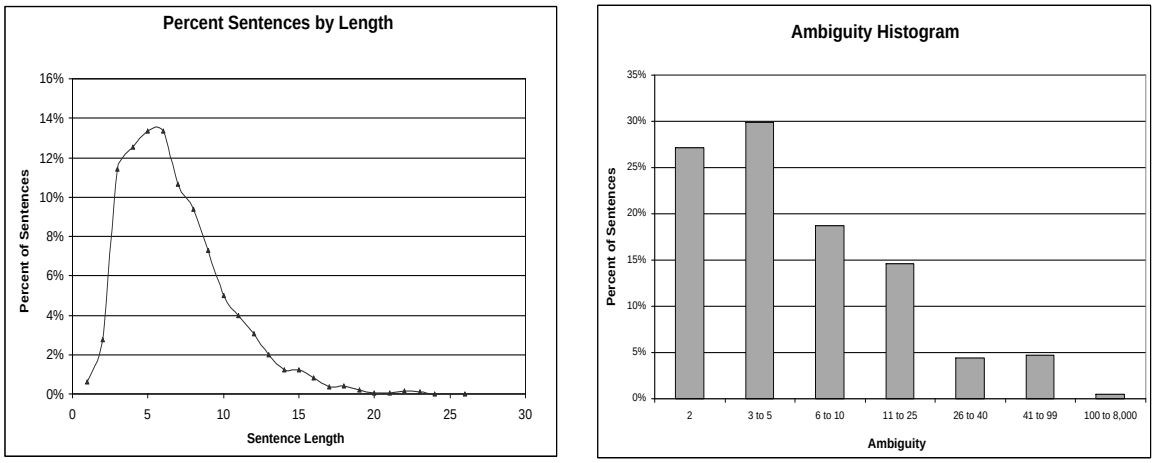
Finally, the HPSG signs provide deep semantic representations for sentences: together with the syntactic analyses of constituents, an underspecified minimal recursion semantics (MRS) representation (Copestake et al., 1999) is built up. This semantic information, unavailable in the Penn Treebank, may provide a useful source of additional features, at least partially orthogonal to syntactic information, for aiding parse disambiguation.

2.2.1 Data Characteristics

The Redwoods corpus sentences are taken from VerbMobil (Wahlster, 2000) transcribed spoken dialogues in the appointment scheduling and travel arrangements domain. Here, we use Version 1.5 of the Redwoods corpus. It dates from June 2002. This version of the data was also used in (Oepen et al., 2002; Toutanova et al., 2002; Toutanova et al., 2003). Newer versions of the corpus exist, the next one being Version 3.0 from August 2003. The experiments reported in (Toutanova et al., 2005b) are performed on Version 3.0.

We list important statistics on the composition and ambiguity of the corpus used here. Figure 2.3 shows the total number of ambiguous sentences, average sentence length, and average number of possible parses per sentence. The total number of sentences in this version of the corpus, including unambiguous ones, is 5307.

Figure 2.4 shows the percentage of sentences by length in 2.4(a) and percentage of sentences by number of possible analyses in 2.4(b).



(a) Length Distribution.

(b) Ambiguity Histogram.

Figure 2.4: Characteristics of the Redwoods corpus Version 1.5. The sentence length distribution and a histogram of numbers of possible analyses are shown.

2.3 Related Work

Most recent machine learning parsing models have been proposed for parsing into Penn Treebank (Marcus et al., 1993) analyses. Additionally, a growing number of models for deeper grammars are being developed. We overview both types of models as they relate to our work. The parsing models differ in the feature representations of syntactic analyses and machine learning methods used. In the discussion here, we will mainly emphasize the feature representations, even though we also specify the learning methods.

2.3.1 Penn Treebank Parsing Models

After generative Probabilistic Context Free Grammars (PCFG) were first applied to Penn Treebank parsing, many ways of improving the feature representation of parsing models have been found. For example, by incorporating more structural context in the features in the form of parent annotation of parse tree nodes, the performance of

PCFG models was greatly improved (Johnson, 1998): labeled precision/recall went up from 73.5/69.7 to 80.0/79.2.

Models including more structural information and lexicalization of each rule with the lexical head, sophisticated smoothing, annotation and rule decomposition schemes led to further improvements in performance (Magerman, 1995; Collins, 1997; Charniak, 2000). The generative model of Charniak (Charniak, 2000) was the best performing one for several years with performance around 90 F-Score.

Klein and Manning (2003) showed that un-lexicalized models, based on markovized annotated context-free rules can perform much better than previously believed. The proposed annotations of parse tree nodes were both internal, in the sense that they are based on structure within a constituent, and external, in the sense that they look at external structure.

Eisner (2002) shows that it is advantageous to use joint representation of all expansions headed by a verb, i.e., to model the sentence (S) and verb phrase (VP) levels jointly, rather than separately. This is relevant to our work because we also model jointly the modifiers and complements of words and take the joint modeling idea even further.

Discriminative models have also been recently applied to Penn Treebank parsing. Such models often have superior performance, due to their properties of more closely optimizing the target accuracy measure and allowing the seamless incorporation of arbitrary non-independent features.

The problem with discriminative models is the computational complexity of training. Because such models need to consider all possible analyses in multiple iterations, training can be prohibitively slow. Various approaches have been proposed to deal with the problem, when the number of possible analyses is too large to deal with explicitly. Examples include re-ranking (Collins, 2000), dynamic programming (Taskar et al., 2004), online learning (Crammer and Singer, 2003) combined with top k re-ranking (McDonald et al., 2005) and approximation techniques (Tsochantaridis et al., 2004).

Currently, the best performing model for Penn Treebank parsing is due to (Charniak and Johnson, 2005). It is a discriminative re-ranking model, which re-ranks the top 50 parse trees according to the generative parser described in (Charniak, 2000). In the terminology of §2.1, it is a linear model which selects a parse tree from among 50 possible ones, proposed by the generative model. Many types of features are incorporated in this model and we only briefly mention the most notable kinds here. In addition to standard features types, such as context-free rules with ancestor annotation and bi-lexical dependencies, the following features are included: a limited number of subtree features, n-gram features connecting the leaves of the tree with l internal nodes, head word projections which achieve joint representation of all rules having the same head, and features measuring parallelism for coordination and heaviness of constituents. The F-Measure of this model is 91 on sentences of length $\leq$ 100.

A different and well-performing representation is the all subtrees representation of Bod (1998). Subtrees of large size and containing many words are incorporated as features and shown to lead to gains in accuracy.

2.3.2 Models for Deep Grammars

Unification-based Grammars

A problem with building generative models for unification-based grammars is that it is difficult to make sure probability mass is not lost to trees which violate the constraints of the grammar. For example, a relative frequency estimator for a probabilistic context-free grammar is not a maximum likelihood estimator for unification-based grammars in the general case. A machine-learning model for unification-based grammars was first proposed in (Abney, 1997). The goal of the work was not to maximize performance but to propose a statistically sound technique for estimation of weighted context-free rule models; the proposed model class was Markov Random Fields. The first model applied to disambiguation for a unification-based grammar on real data was proposed in (Johnson et al., 1999). It is a discriminative log-linear model in the

terminology of §2.1. The proposed features are specific to LFG, the grammar formalism of the data set used. In addition to local context-free rule features, there are features indicating grammatical function, high and low attachment, parallelism for coordinates structures, and f-structure atomic attribute-value pairs. A similar model was applied to a larger corpus containing LFG parses of Penn Treebank trees (Riezler et al., 2002).

Previous work on HPSG for the Redwoods corpus includes our work (Oepen et al., 2002; Toutanova and Manning, 2002; Toutanova et al., 2003; Toutanova et al., 2005b) and work by other researchers (Baldridge and Osborne, 2003; Osborne and Baldbridge, 2004; Baldridge and Osborne, 2004). We studied generative and discriminative models for HPSG parsing, the generative models being deficient and inconsistent. We found that discriminative models consistently outperformed corresponding generative models (Toutanova et al., 2005b). We applied context-free rule based models with lexicalization and ancestor annotation to derivation trees, and combined these with models using semantic representations. The best result we obtained via combining features derived from multiple representations of HPSG signs was 82.7% on Version 1.5 of the Redwoods treebank. Later we will show that the present models outperform the previous ones considerably. The model most relevant to our work is the one described in (Osborne and Baldbridge, 2004); it is a voted combination of several log-linear models based on different parts of the representations available in the Redwoods corpus. The emphasis of that work was on active learning and not so much on developing good representations for HPSG analyses. The features were local rule features with ancestors, and features form the semantic representations, similar to our features in (Toutanova et al., 2005b). The voted model achieved an accuracy of 84.2% on Version 1.5 of Redwoods. We show that our present model outperforms this model as well.

Miyao and Tsujii (2005) propose a log-linear model for HPSG parsing, which they apply to parsing Penn Treebank sentences. The group constructed a corpus of HPSG parses semi-automatically using the existing parses in the Penn Treebank (Miyao et al., 2004). The model used has only features based on lexicalized local rules, including counting the number of words spanned, distance between a head and dependent word,

and POS tags and lexical entry types for these words. Because the grammar is wide coverage and highly ambiguous, two techniques are applied to deal with the computational complexity issues: pre-filtering by limiting the possible lexical types and dynamic programming to compute feature expectations on a packed feature forest.

Other Deep Grammars

Statistical models have also been defined for other deep grammars, such as Lexicalized Tree-Adjoining Grammar (Srinivas and Joshi, 1999) and Combinatory Categorial Grammar (CCG) (Steedman, 1996). Such grammars and corpora for them have been developed semi-automatically using the annotated Penn Treebank corpus (Xia, 1999; Chen and Vijay-Shanker, 2000; Hockenmaier and Steedman, 2002a). A generative model for CCG, using context-free rule expansions with lexicalization, grand-parent annotation, distance measures and lexicalization was proposed in (Hockenmaier and Steedman, 2002b), and found to perform well. The model was further improved in (Hockenmaier, 2003b) to model word-word dependencies in the underlying predicate-argument structure. A discriminative log-linear model for CCG was proposed in (Clark and Curran, 2004), achieving better performance.

2.3.3 Kernels for Parse Trees

An example of a kernel used for natural language parsing is the all subtrees kernel of Collins and Duffy (2001). This kernel effectively implements the all-subtrees representation of parse trees (Bod, 1998) by the application of a fast dynamic programming algorithm for computing the number of common subtrees of two trees. The features correspond to arbitrary subtrees of a parse tree, with the restriction that, if any children of a node are included in the subtree, then all children of that node are included. Collins and Duffy (2002) reported an experiment using this kernel to re-rank the top n parse trees produced by the generative parser of Collins (1999). The machine learning method used was a voted perceptron. The improvement in average precision and recall was 0.6 absolute (from 88.2 to 88.8) on sentences of length ≤ 100 words, which is a 5.1% relative reduction in error. Shen and Joshi (2003) achieved a similar result using an SVM rather than a voted perceptron and using the same tree

kernel; however, they trained on only about a quarter of the training set because of computational complexity.

Overall, the tree kernel has not been shown to have superior performance to context-free rule based representations. For example, the re-ranking model of (Collins, 1999), using boosting and a feature representation based on local rule features plus additional lexicalization, achieves better performance – 89.6 recall and 89.9 precision on sentences of length ≤ 100. A reason for this could be the large space of possible features, which may result in overfitting. In Collins and Duffy (2001), larger subtrees are discounted and an it does not hurt to limit the maximum depth of the used subtrees to a small number.

Another tree kernel, more broadly applicable to Hierarchical Directed Graphs, was proposed in (Suzuki et al., 2003). Many other interesting kernels have been devised for sequences and trees, with application to sequence classification and parsing. A good overview of kernels for structured data can be found in (Gärtner et al., 2002).

2.4 Our Approach

In line with previous work on unification-based grammars, we build discriminative models for parse disambiguation. Because the number of analyses is quite limited, as seen from the corpus statistics in §2.2, we do not need to worry about computational complexity when defining the features of our models.

Compared to the usual notion of discriminative models (placing classes on rich observed data) discriminative PCFG parsing with plain context-free rule features may look naive, since most of the features (in a particular tree) make no reference to observed input at all. Lexicalization, which puts an element of the input on each tree node, so all features do refer to the input, addresses this problem to some extent, but this may not be sufficient. This is even more true for an HPSG grammar, because it is very lexicalized. The node labels in derivation trees indicate the type of relationship of the head and non-head constituents which join at each node. Therefore it is worth exploring feature representations which are word-centered and depart to a larger

extent from the context-free rule paradigm.

We propose a linguistically motivated representation centered around lexicalized broad structural fragments. This representation naturally extends to defining tree kernels, by building on previous work on string kernels. By using kernels, we effectively include novel types of features, previously unexplored for parsing.

We represent parse trees as lists of paths (*leaf projection paths*) from words to the top level of the tree, which includes both the head-path (where the word is a syntactic head) and the non-head path. This allows us to capture, for example, cases of non-head dependencies which were also discussed by (Bod, 1998) and were used to motivate large subtree features, such as "more careful than his sister" where "careful" is analyzed as head of the adjective phrase, but "more" licenses the "than" comparative clause. This representation of trees as lists of projection paths (strings) allows us to explore string kernels on these paths and combine them into tree kernels.

We use SVM models for parse selection, as they are a high-performance machine learning method and allow for the use of kernels. We reviewed the formulation of SVM models in the parse selection setting in §2.1.1.

2.5 Proposed Representation

In this section, after describing in more detail the form of syntactic analyses of HPSG used, we introduce the representation of parse trees as leaf projection paths.

2.5.1 Representing HPSG Signs

As mentioned in §2.2, we have not used the complete HPSG signs for defining statistical disambiguation models. The main representation for our models is the derivation tree representation, shown in Figure 2.2(a). This representation has also been used in previous work on the Redwoods corpus (Toutanova et al., 2002; Toutanova and Manning, 2002; Osborne and Baldbridge, 2004). Another example of a derivation

tree, for the sentence *Let us plan on that*, is shown in Figure 2.5.[3]

Additionally, we annotate the nodes of the derivation trees with information extracted from the HPSG sign. The annotation of nodes is performed by extracting values of feature paths from the feature structure or by propagating information from children or parents of a node. In theory with enough annotation at the nodes of the derivation trees, we can recover the whole HPSG signs.

Here we describe three node annotations that proved very useful for disambiguation. One is annotation with the values of the feature path synsem.local.cat.head – its values are basic parts of speech such as *noun, verb, prep, adj, adv*. Another is phrase structure category information associated with the nodes, which summarizes the values of several feature paths and is available in the Redwoods corpus as Phrase-Structure trees (an example Phrase-Structure tree is shown in Figure 2.2(b)). The third is annotation with lexical type (*le-type*), which is the type of the head word at a node. The preterminals in Figure 2.5 are lexical item identifiers — identifiers of the lexical entries used to construct the parse. The *le-types* are about 500 types in the HPSG type hierarchy and are the direct super-types of the lexical item identifiers. The le-types are not shown in this figure, but can be seen at the leaves in Figure 2.6. For example, the lexical type of **LET_V1** in the figure is *v_sorb*, a subtype of subject to object raising verbs. In Figure 2.5, the only annotation shown is with the values of synsem.local.cat.head.

2.5.2 The Leaf Projection Paths View of Parse Trees

We define our representation of parse trees, applied to derivation trees.

The *projection path* of a leaf is the sequence of nodes from the leaf to the root of the tree. In Figure 2.6, the *leaf projection paths* for three of the words are shown.

We can see that a node in the derivation tree participates in the projection paths of all words dominated by that node. The original local rule configurations — a node and its children, do not occur jointly in the projection paths; thus, if special

[3]This sentence has three possible analyses depending on the attachment of the preposition "on" and whether "on" is an adjunct or complement of "plan".

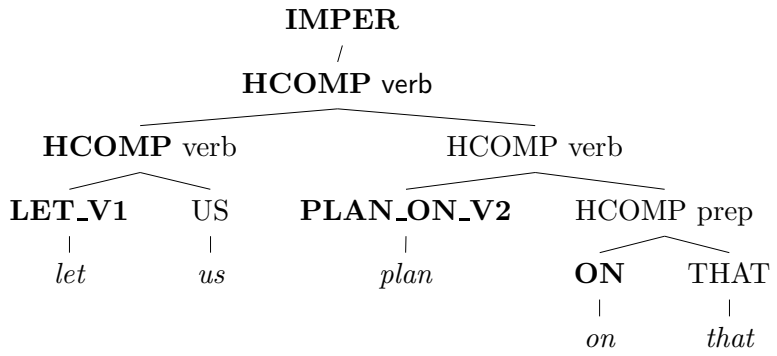


Figure 2.5: Derivation tree for the sentence *Let us plan on that.* The head child of each node is shown in bold.

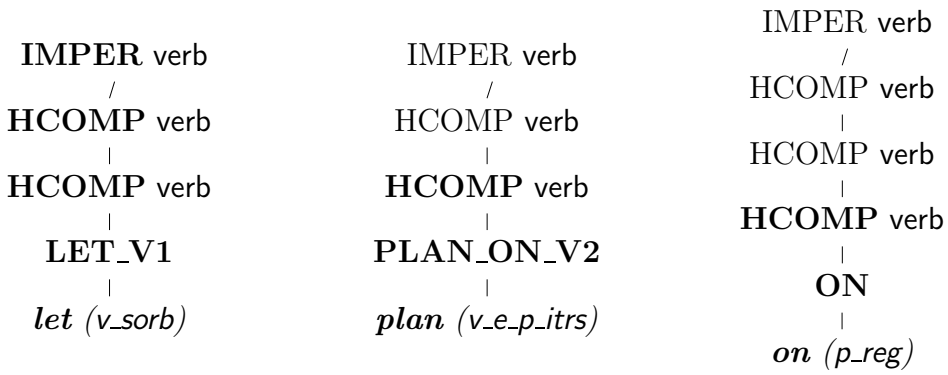


Figure 2.6: Paths to top for three leaves. The nodes in bold are head nodes for the leaf word and the rest are non-head nodes.

annotation is not performed to recover it, this information is lost.

As seen in Figure 2.6, and as is always true for a grammar that produces non-crossing lexical dependencies, there is an initial segment of the projection path for which the leaf word is a syntactic head (called *head path* from here on), and a final segment for which the word is not a syntactic head (called *non-head path* from here on).

If, in a traditional parsing model that estimates the likelihood of a local rule expansion given a node (such as e.g. (Collins, 1997)), the tree nodes are annotated with the word of the lexical head, some information present in the word projection paths can be recovered. However, this is only information in the head projection paths,

and it is broken across multiple bar levels. Other models, such as (Eisner, 2002) and (Charniak and Johnson, 2005) use explicit representation of complete maximal head projections, and thus model jointly complete head paths. In further experiments we show that the non-head part of the projection path is very helpful for disambiguation, and that the head and non-head paths can be modelled better by the application of kernels.

Using this representation of derivation trees, we apply string kernels to the leaf projection paths and combine those to obtain kernels on trees. In the rest of this chapter we explore the application of string kernels to this task, comparing the performance of the new models to models using more standard rule features.

2.6 Tree and String Kernels

2.6.1 Kernels on Trees Based on Kernels on Projection Paths

So far we have defined a representation of parse trees as lists of strings corresponding to projection paths of words. Now we formalize this representation and show how string kernels on projection paths extend to tree kernels.

We introduce the notion of a keyed string — a string that has a key, which is some letter from an alphabet Σ_{Keys}. We can denote a keyed string by a pair (a, x), where $a \in \Sigma_{Keys}$ is a key, and x is a string. In our application, the strings x are representations of head and non-head word projection paths, and the keys w are words. More specifically, for every word w, if x_h is the string representation of the head projection path of w as a sequence of annotated derivation tree nodes, and x_{nh} is the string representation of the non-head projection path of w, then the keyed strings $({}^{h}w, x_h)$ and $({}^{nh}w, x_{nh})$ represent the two parts of the word projection path of w. The keys have the form ${}^{loc}w$ where *loc* (from location) indicates whether the keyed string represents a head path ($loc = h$) or a non-head path ($loc = nh$). Additionally, for reducing sparsity, for each keyed string $({}^{loc}w, x_{loc})$, we also include a keyed string $({}^{loc}le_w, x_{loc})$, where le_w is the le-type of the word w. Thus each projection path occurs

twice in the list representation of the tree – once keyed by the word, and once by its le-type. For example, for the word *let* in Figure 2.6, the head path of the word is x_h="*SOP LET_V1:verb HCOMP:verb HCOMP:verb IMPER:verb EOP*", the non-head path is x_{nh}="*SOP EOP*", and the le-type of *let* is *v_sorb*. Here the symbols *SOP* and *SOP* represent start and end of path. This information about the word *let* will be represented in the form of keyed strings as follows:

(^{h}*let*, *SOP LET_V1:verb HCOMP:verb HCOMP:verb IMPER:verb EOP*)

(^{nh}*let*, *SOP EOP*)

(^{h}*v_sorb*, *SOP LET_V1:verb HCOMP:verb HCOMP:verb IMPER:verb EOP*)

(^{nh}*v_sorb*, *SOP EOP*)

For a given kernel K_p on strings, we define its extension to keyed strings as follows: $K((a,x),(b,y)) = K_p(x,y)$, if $a = b$, and $K((a,x),(b,y)) = 0$, otherwise. We use this construction for all string kernels applied in this work. It is easy to show that if K_p is a kernel, then K is a kernel as well, using a tensor product of the kernels $K_\delta(a,b)$ and $K_p(x,y)$, where $K_\delta(a,b)$ is defined as follows: $K_\delta(a,b) = 1$ if $a = b$, and $K_\delta(a,b) = 0$, otherwise (Berg et al., 1984).

Given a tree $t_1 = ((a_1,x_1),\ldots,(a_n,x_n))$ and a tree $t_2 = ((b_1,y_1),\ldots,(b_m,y_m))$, and a kernel K on keyed strings, we define a kernel KT on trees as follows:

$$KT(t_1,t_2) = \sum_{i=1}^{n}\sum_{j=1}^{m} K((a_i,x_i),(b_j,y_j))$$

This can be viewed as a convolution kernel (Haussler, 1999) and therefore KT is a valid kernel (positive definite symmetric), if K is a valid kernel. In particular, following the definitions in §2.1.2, we can define KT as an R-convolution with respect to the following relation R: $R(i,(a,x),t)$ is true, iff (a,x) is the i-th keyed string in the representation of the tree t. KT is an R-convolution of K and the constant unity kernel $K_1(i,j) = 1$, for all $i,j \in N$.

2.6.2 String Kernels

We experimented with some of the string kernels proposed in (Lodhi et al., 2000; Leslie and Kuang, 2003), which have been shown to perform very well for indicating string similarity in other domains. In particular we applied the N-gram kernel, Subsequence kernel, and Wildcard kernel. We refer the reader to (Lodhi et al., 2000; Leslie and Kuang, 2003) for detailed formal definition of these kernels, and restrict ourselves to an intuitive description here. In addition, we devised two new kernels – the Repetition kernel, and the Distance-weighted n-gram kernel, which we describe in detail.

The n-gram kernel has been used widely in previous work on parse disambiguation, even if it has not been explicitly given this name. For example, when order n markovization is used for local rule features, this is an implicit use of an n-gram string kernel. The other kernels we introduce have not been commonly used for parsing.

The kernels used here can be defined as the inner product of appropriately defined feature vectors for the two strings $K(x, y) = \langle \Phi(x), \Phi(y) \rangle$. For most kernels, there exists a feature map Φ which maps the strings into a finite dimensional Euclidean space. We will define all kernels, expect for the new Distance-weighted n-gram kernel, using such a map into a finite dimensional vector space. We will define the Distance-weighted n-gram kernel through a convolution and not through an explicit map (for convenience and because we would need to use an infinite dimensional space to define it otherwise).

The maps $\Phi(x)$ defining a string kernel will be maps from all finite sequences from a string alphabet Σ to a Euclidean space R^m, indexed by an appropriately defined set of subsequences from Σ. As a simple example, the 1-gram string kernel maps each string $x \in \Sigma^*$ to a vector with dimensionality $|\Sigma|$ and each element in the vector indicates the number of times the corresponding symbol from Σ occurs in x. For example, the head path of *let* in Figure 2.6 is represented by the string x="*SOP LET_V1:verb HCOMP:verb HCOMP:verb IMPER:verb EOP*" . The symbols *SOP* and *EOP* represent start and end of path, respectively. Each dimension in $\Phi(x)$ is indexed by a letter of the alphabet of the string, namely, the set of annotated node labels. If the letters are indexed in alphabetical order, the feature representation of x will be

the following (only the non-zero valued coordinates are shown):

α	$\Phi_\alpha(x)$
EOP	1
HCOMP:verb	2
IMPER	1
LET_V1:verb	1
SOP	1

Figure 2.7: Feature map for 1-gram kernel for the string x="*SOP LET_V1:verb HCOMP:verb HCOMP:verb IMPER:verb EOP*".

Repetition Kernel

The *Repetition kernel* is similar to the 1-gram kernel. It improves on the 1-gram kernel by better handling cases with repeated occurrences of the same symbol. Intuitively, in the context of our application, this kernel captures the tendency of words to take (or not take) repeated modifiers of the same kind. For example, it may be likely for a certain verb to take one PP-modifier, but less likely for it to take two or more.

More specifically, the *Repetition kernel* is defined such that its vector space is indexed by all sequences from Σ composed of the same symbol. The feature map obtains matching of substrings of the input string to features, allowing the occurrence of gaps. There are two discount parameters λ_1 and λ_2. λ_1 serves to discount features for the occurrence of gaps, and λ_2 discounts longer symbol sequences.

Formally, for an input string x, the value of the feature vector for the feature index sequence $\alpha = a \ldots a$, $|\alpha| = k$, is defined as follows: Let s be the left-most minimal contiguous substring of x that contains α, $s = s_1 \ldots s_l$, where for indices $i_1 = 1, i_2, \ldots i_k = l$, $s_{i_1} = a = s_{i_2} = \ldots = s_{i_k}$. Then $\Phi_\alpha^{Repetition}(x) = \lambda_1^{l-k}\lambda_2^k$.

For example, if $\lambda_1 = .5, \lambda_2 = .3$, $\Phi_a(abcadaf) = .3$, $\Phi_{aa}(abcadaf) = .3 \times .3 \times .5 \times .5 = 0.0225$, and $\Phi_{aaa}(abcadaf) = .3 \times .3 \times .3 \times .5 \times .5 \times .5 = 0.003375$. For our previous example with the head path of *let*, its feature representation for $\lambda_1 = .5, \lambda_2 = .3$ is shown in Figure 2.8.

α	$\Phi_\alpha(x)$
EOP	0.3
HCOMP:verb	0.3
HCOMP:verb HCOMP:verb	0.09
IMPER	0.3
LET_V1:verb	0.3
SOP	0.3

Figure 2.8: Feature map for Repetition kernel for the string x="*SOP LET_V1:verb HCOMP:verb HCOMP:verb IMPER:verb EOP*", $\lambda_1 = .5, \lambda_2 = .3$.

Distance-weighted N-gram Kernel

In estimating the probability of occurrence of complements and modifiers given a head, the distance of the those modifiers has been shown to be very useful. This is evidenced, for example, in the work of (Collins, 1997), for estimating the probability of lexicalized rule expansions. However, the addition of a distance feature with D values introduces some additional sparsity and the gain from the additional information may not always be large enough to outweigh it.

Here we propose a different way of taking distance into account by a distance-weighted n-gram kernel. This method for distance weighting can also be incorporated in many other string kernels such as the Wildcard and Subsequence kernels which we describe shortly. For concreteness, we will describe our definition of a distance-weighted 1-gram kernel. The extension to higher k-grams and other kernels will be obvious. The 1-gram kernel for strings $a_1, a_2, \ldots, a_n$ and $b_1, b_2, \ldots, b_m$ can be represented as a convolution kernel in the following manner. Let R be a four-place relation on strings defined as follows: $(x_0, x_1, x_2, x) \in R$ iff $x_0 \circ x_1 \circ x_2 = x$, and $|x_1| = 1$. Here $\circ$ denotes string concatenation and $|x|$ denotes the length of x.

The 1-gram string kernel can be defined as an R-convolution of the kernels on strings $K_1(x, y) = 1\ \forall x, y$, $K_2(x, y) = 1$ iff $x = y$ and 0 otherwise, and $K_3(x, y) = K_1(x, y)$. The 1-gram kernel is defined as the zero extension of the generalized convolution: $K(x, y) = \sum_{R(x_0,x_1,x_2,x)} \sum_{R(y_0,y_1,y_2,y)} K_1(x_0, y_0) \times K_2(x_1, y_1) \times K_3(x_2, y_2)$. This directly follows the definitions in §2.1.2.

To obtain a distance weighted 1-gram kernel we propose to substitute the constant kernels K_1 and/or K_3 by a Gaussian kernel on the lengths of the strings. $K_1(x, y) = e^{(-(d_x - d_y)^2/\sigma^2)}$, where $d_x = |x|$ and $d_y = |y|$. Thus the common 1-grams which occur an equal distance form the beginning in the two strings will contribute most weight to the similarity, and 1-grams that occur at hugely different locations will contribute much less.

By using the convolution kernel representation, we can see how the distance weighting extends to many other string kernels. Additionally, it can be adapted to tree kernels to compare the depths at which common subtrees occur or the weights of constituents.

Previously Proposed Kernels

The weighted *Wildcard kernel* performs matching by permitting a restricted number of matches to a wildcard character. A (k, m) wildcard kernel has as feature indices k-grams with up to m wildcard characters. Any character matches a wildcard. For example the 3-gram *aab* will match the feature index $a * b$ in a (3,1) wildcard kernel. The weighting is based on the number of wildcard characters used – the weight is multiplied by a discount λ for each wildcard.

As a specific example, Figure 2.9 shows the feature representation of the head path of *let* in (2,2) wildcard card with $\lambda = .5$. The feature index strings are bi-grams with up to 2 wildcards.

The *Subsequence kernel* was defined in (Lodhi et al., 2000). We used a variation where the kernel is defined by two integers (k, g) and two discount factors λ_1 and λ_2 for gaps and characters. A subseq(k,g) kernel has as features all n-grams with $n \leq k$. The g is a restriction on the maximal span of the n-gram in the original string – e.g. if $k = 2$ and $g = 4$, the two letters of a 2-gram can be at most $g - k = 2$ letters apart in the original string. The weight of a feature is multiplied by λ_1 for each gap, and by λ_2 for each non-gap. For example, if $\lambda_1 = .5, \lambda_2 = 3, k = 2, g = 3$, $\Phi_{aa}(abcadaf) = 3 \times 3 \times .5 = .45$. The feature index *aa* matches only once in the string with a span at most 3 – for the sequence *ada* with 1 gap.

α	$\Phi_\alpha(x)$
**	0.25
* EOP	0.5
* HCOMP:verb	1.0
* IMPER:verb	0.5
* LET_V1:verb	0.5
HCOMP:verb *	1.0
HCOMP:verb HCOMP:verb	1.0
HCOMP:verb IMPER:verb	1.0
IMPER:verb *	0.5
IMPER:verb EOP	1.0
LET_V1:verb *	0.5
LET_V1:verb HCOMP:verb	1.0
SOP *	0.5
SOP LET_V1:verb	1.0

Figure 2.9: Feature map for a (2,2) Wildcard kernel for the string x="*SOP LET_V1:verb HCOMP:verb HCOMP:verb IMPER:verb EOP*", $\lambda = .5$.

Figure 2.10 shows the feature representation of the head path of *let* according to a (2,3) Subsequence kernel with $\lambda_1 = .5, \lambda_2 = 3$. The value of the feature SOP is $3 = \lambda_2$, because there is one letter and no gaps. The value of the feature HCOMP:verb IMPER is $x = \lambda_2 \times \lambda_2 + \lambda_2 \times \lambda_2 \times \lambda_1 = 13.5$, because this feature occurs twice in the head path – once without gaps, and once with one gap.

The details of the algorithms for computing the kernels can be found in the aforementioned papers (Lodhi et al., 2000; Leslie and Kuang, 2003). To summarize, the kernels can be implemented efficiently using tries.

2.7 Experiments

In this section we describe our experimental results using different string kernels and different feature annotation of parse trees. We performed experiments using the version of the Redwoods corpus described in §2.2.1. We discarded the unambiguous sentences from the training and test sets. All models were trained and tested using

α	$\Phi_\alpha(x)$
EOP	3.0
HCOMP:verb	6.0
HCOMP:verb EOP	4.5
HCOMP:verb HCOMP:verb	9.0
HCOMP:verb IMPER:verb	13.5
IMPER	3.0
IMPER:verb EOP	9.0
LET_V1:verb	3.0
LET_V1:verb HCOMP:verb	13.5
SOP	3.0
SOP HCOMP:verb	4.5
SOP LET_V1:verb	9.0

Figure 2.10: Feature map for a (2,3) Subsequence kernel for the string x="*SOP LET_V1:verb HCOMP:verb HCOMP:verb IMPER:verb EOP*", $\lambda_1 = .5, \lambda_2 = 3$.

10-fold cross-validation. The i-th fold, consisting of training set $train_i$ and test set $test_i$, for $i = 1, \ldots, 10$, was generated by placing every $(10k + i)$-th sentence in $test_i$ and the remaining sentence in $train_i$. Accuracy results are reported as percentage of sentences where the correct analysis was ranked first by the model.

2.7.1 SVM Implementation

We learn Support Vector Machine (SVM) ranking models using the software package SVM^{light} (Joachims, 1999). We also normalized the kernels:

$$K'(t_1, t_2) = \frac{K(t_1,t_2)}{\sqrt{K(t_1,t_1)}\sqrt{K(t_2,t_2)}}.$$

For kernels that can be defined through relatively small feature maps Φ it is advantageous to explicitly generate feature vectors for all examples and to use SVM^{light} with a linear kernel. This is because it is likely the special optimization techniques for linear kernels will make the computation faster. This is what we did for all kernels implemented here, with the exception of the Distance-weighted n-gram kernel, which requires an infinite dimensional feature map. After creating feature vectors for all

trees $\Phi(t_j^i)$, we learned preference ranking SVMs (Joachims, 2002) with a linear kernel. The ranking specified gave highest rank to the correct analysis of each sentence, and lower equal ranks to all other analyses. This results in the same optimization problem formulation as outlined in §2.1.1.

At the time the experiments were performed, SVM^{light} did not contain implementation of non-linear kernels for preference ranking problems. The current version does have this feature and additionally, the SVM^{struct} extension of SVM^{light} should also provide capabilities for solving the kernelized version of the optimization problem in §2.1.1.

Because these options were not available, we solved the optimization problem through representing it as an equivalent binary classification problem, as described in (Shen and Joshi, 2003). The trick is to have pairs of correct and incorrect analyses be represented as single positive or negative instances. In particular, Shen and Joshi (2003) propose to add a positive example for every ordered pair $[t_0^i, t_j^i]$ for every training sentence i and incorrect analysis t_j^i, $j > 0$, and a negative example for every reversed pair $[t_j^i, t_0^i]$. Shen and Joshi (2003) also show how to extend a kernel on parse trees to an equivalent kernel on pairs of parse trees of this form and we use their method. We found that we can cut the size of the training set in half, by adding only positive examples – for the pairs $[t_0^i, t_j^i]$ – and constraining the intercept b to $b = 0$. It is easy to show that this results in an optimization problem equivalent to the one in §2.1.1.

The structure of the rest of the experiments section is as follows. First we describe the results of a controlled experiment using a limited number of features, and aimed at comparing models using local rule features to models using leaf projection paths in §2.7.2. Next we describe models using more sophisticated string kernels on projection paths in §2.7.3.

2.7.2 The Leaf Projection Paths View versus the Context-Free Rule View

In order to evaluate the gains from the new representation, we describe the features of three similar models, one using leaf projection paths, and two using derivation tree rules. Additionally, we train a model using only the features from the *head-path* parts of the projection paths to illustrate the gain of using the *non-head* path. As we will show, a model using only the head-paths has almost the same features as a rule-based tree model.

All models here use derivation tree nodes annotated with only the rule schema name as in Figure 2.5 and the synsem.local.cat.head value. We will define these models by their feature map from trees to vectors. It will be convenient to define the feature maps for all models by defining the set of features through templates. The value $\Phi_\alpha(t)$ for a feature α and tree t, will be the number of times α occurs in the tree. It is easy to show that the kernels on trees we introduce in §2.6.1, can be defined via a feature map that is the sum of the feature maps of the string kernels on projection paths.

As a concrete example, for each model we show all features that contain the node "HCOMP:verb" from Figure 2.5, which covers the phrase *plan on that*.

Bi-gram Model on Projection Paths (2PP)

The features of this model use a projection path representation, where the keys are not the words, but the *le-types* of the words. The features of this model are defined by the following template : $({}^{loc}leType, node_i, node_{i+1})$. The *loc* superscript indicates whether this feature matches a head (h) or a non-head (nh) path as in our definitions for keyed string representations of projection paths of §2.6.1, $leType$ is the le-type of the path leaf, and $node_i, node_{i+1}$ is a bi-gram from the path.

The node "HCOMP:verb" is part of the head-path for *plan*, and part of the non-head path for *on* and *that*. The le-types of the words *let, plan, on*, and *that* are, with abbreviations, *v_sorb, v_e_p, p_reg*, and *n_deic_pro_sg* respectively. In the following

examples, the node labels are abbreviated as well; As before, EOP is a special symbol for end of path and SOP is a special symbol for start of path. Therefore the features that contain the node will be:

(^{h}v_e_p,PLAN_ON:verb,HCOMP:verb)

(^{h}v_e_p,HCOMP:verb,EOP)

(^{nh}p_reg,SOP,HCOMP:verb)

(^{nh}p_reg,HCOMP:verb,HCOMP:verb)

(^{nh}n_deic_pro_sg,HCOMP:prep,HCOMP:verb)

(^{nh}n_deic_pro_sg,HCOMP:verb,HCOMP:verb)

Bi-gram Model on Head Projection Paths (2HeadPP)

This model has a subset of the features of Model 2PP — only those obtained by the *head path* parts of the projection paths. For our example, it contains the subset of features of 2PP that have location ^{h}, which are only the following:

(^{h}v_e_p,PLAN_ON:verb,HCOMP:verb)

(^{h}v_e_p,HCOMP:verb,EOP)

Rule Tree Model I (Rule I)

The features of this model are defined by the two templates: $(^{h}leType, node, child_1, child_2)$ and $(^{nh}leType, node, child_1, child_2)$. The location superscript on $leType$ is an indication of whether the tuple contains the le-type of the head or the non-head child as its first element. The features containing the node "HCOMP:verb" are ones from the expansion at that node and also from the expansion of its parent:

(^{h}v_e_p,HCOMP:verb,PLAN_ON:verb,HCOMP:prep)

(^{nh}p_reg,HCOMP:verb,PLAN_ON:verb,HCOMP:prep)

(^{h}v_sorb,HCOMP:verb,HCOMP:verb,HCOMP:verb)

(^{nh}v_e_p,HCOMP:verb,HCOMP:verb,HCOMP:verb)

Rule Tree Model II (Rule II)

This model splits the features of model Rule I in two parts, to mimic the features of the projection path models. It has features from the following templates:

$(^{h}leTypeHead, node, headChild)$

$(^{nh}leTypeNonHead, node, nonHeadChild)$

The location indicator shows again whether the tuple contains the le-type of the head or the non-head child and additionally, whether the child node included is the head or the non-head child. The features containing the "HCOMP:verb" node are:

(^{h}v_e_p,HCOMP:verb,PLAN_ON:verb)

(^{nh}p_reg,HCOMP:verb,HCOMP:prep)

(^{nh}v_e_p,HCOMP:verb,HCOMP:verb)

This model has less features than model Rule I, because it splits each rule into its head and non-head parts and does not have the two parts jointly. We can note that this model has all the features of 2HeadPP, except the ones involving start and end of path, due to the first template. The second template leads to features that are not even in 2PP because they connect the head and non-head paths of a word, which are represented as separate strings in 2PP.

Overall, we can see that models Rule I and Rule II have the information used by 2HeadPP (and some more information), but do not have the information from the non-head parts of the paths in Model 2PP. Figure 1 shows the average parse ranking accuracy obtained by the four models as well as the number of features used by each model. Model Rule I did not do better than model Rule II, which shows that joint representation of rule features was not very important. The large improvement of 2PP over 2HeadPP (13% error reduction) shows the usefulness of the non-head projection paths. The error reduction of 2PP over Rule I is also large – 9% error reduction. Further improvements over models using rule features were possible by considering more sophisticated string kernels and word keyed projection paths, as will be shown in the following sections.

Model	Features	Accuracy
2PP	36,623	82.70
2HeadPP	11,490	80.14
Rule I	28,797	80.99
Rule II	16,318	81.07

Figure 2.11: Accuracy of models using the leaf projection path and rule representations.

No.	Name	Example
0	Node Label	*HCOMP*
1	synsem.local.cat.head	*verb*
2	Label from Phrase Struct Tree	*S*
3	Le Type of Lexical Head	*v_sorb_le*
4	Lexical Head Word	*let*

Figure 2.12: Annotated features of derivation tree nodes. The examples are from one node in the head path of the word *let* in Figure 2.5.

2.7.3 Experimental Results using String Kernels on Projection Paths

In the present experiments, we have limited the derivation tree node annotation to the features listed in Figure 2.12. Many other features from the HPSG signs are potentially helpful for disambiguation, and incorporating more useful features would be a direction in which to extend the current work. However, given the size of the corpus, a single model cannot usefully profit from a huge number of features. Previous work (Toutanova and Manning, 2002; Toutanova et al., 2002; Osborne and Baldbridge, 2004) has explored combining multiple classifiers using different features. We report results from such an experiment as well.

Using Node Label and Head Category Annotations

The simplest derivation tree node representation that we consider consists of features 0 and 1 – schema name and category of the lexical head. All experiments in this subsection are performed using this derivation tree annotation. We briefly mention

Kernel	**Features**	**Accuracy**
1-gram	44,278	82.21
Repetition	52,994	83.59
2-gram	104,331	84.15

Figure 2.13: Comparison of the Repetition kernel to 1-gram and 2-gram.

Kernel	Accuracy
1-gram simple	82.08
1-gram weighted	82.92

Figure 2.14: Comparison of the un-weighted and distance-weighted 1-gram.

results from the best string kernels when using other node annotations, as well as a combination of models using different features in the following subsection.

To evaluate the usefulness of our Repetition Kernel, defined in §2.6.2, we performed several simple experiments. We compared it to a 1-gram kernel, and to a 2-gram kernel. The results – number of features per model and accuracy – are shown in Figure 2.13. The models shown in this figure include both features from projection paths keyed by words and projection paths keyed by le-types. The results show that the Repetition kernel achieves a noticeable improvement over a 1-gram model (7.8% error reduction), with the addition of only a small number of features. In most projection paths, repeated symbols will not occur and the Repetition kernel will behave like a 1-gram kernel for the majority of cases. The additional information it captures about repeated symbols gives a sizable improvement. The bi-gram kernel performs better but at the cost of the addition of many features. It is likely that for large alphabets and small training sets, the Repetition kernel may outperform the bi-gram kernel.

We compared the distance weighted 1-gram kernel to the un-weighted 1-gram. Results are shown in Figure 2.14.[4] The gain in accuracy is .84 absolute and the relative reduction in error is 4.7%. This result is quite encouraging considering that the only additional information this model is using compared to a simple 1-gram kernel

[4]For these experiments, the kernels were un-normalized. This accounts for the difference in accuracy of the 1-gram kernel in Figure 2.13 and the 1-gram kernel in this figure.

is the distance between matching symbols. This approach is promising for modeling many other numerical properties of syntactic analyses, such as weight of constituents, distance between head words and dependents, and constituent parallelism for coordination. By varying the variance σ of the weighting kernel, we can achieve multiple resolutions of distance-weighting.

We do not use the Distance-weighted kernel in the rest of the experiments in this section. For computational reasons, we could not incorporate it in more feature-rich models. In short, the problem was storing the kernel matrix in memory which required extensive engineering. However, it would certainly be very interesting to see its performance in more sophisticated models and this is a subject of future research.

From this point on, we will fix the string kernel for projection paths keyed by words — it will be a linear combination of a bi-gram kernel and a Repetition kernel. We found that, because lexical information is sparse, going beyond 2-grams for lexically headed paths was not useful. The projection paths keyed by le-types are much less sparse, but still capture important sequence information about the syntactic frames of words of particular lexical types.

To study the usefulness of different string kernels on projection paths, we first tested models where only le-type keyed paths were represented, and then tested the performance of the better models when word keyed paths were added (with a fixed string kernel that interpolates a bi-gram and a Repetition kernel).

Figure 2.15 shows the accuracy achieved by several string kernels as well as the number of features (in thousands) they use. As can be seen from the figure, the models are very sensitive to the discount factors used. Many of the kernels that use some combination of 1-grams and possibly discontinuous bi-grams performed at approximately the same accuracy level, for example, the $\mathsf{wildcard}(2,1,\lambda)$ and $\mathsf{subseq}(2,g,\lambda_1,\lambda_2)$ kernels. Kernels that use 3-grams have many more parameters, and even though they can be marginally better when using le-types only, their advantage when adding word keyed paths disappears. A limited amount of discontinuity in the Subsequence kernels was useful. Overall Subsequence kernels were slightly better than Wildcard kernels. The major difference between the two kinds of kernels is that the Subsequence kernel

Model	Features		Accuracy	
	le	w & le	le	w & le
1gram	13K	-	81.43	-
2gram	37K	141K	82.70	84.11
wildcard (2,1,.7)	62K	167K	83.17	83.86
wildcard (2,1,.25)	62K	167K	82.97	-
wildcard (3,1,.5)	187K	291K	83.21	83.59
wildcard (3,2,.5)	220K		82.90	-
subseq (2,3,.5,2)	81K	185K	83.22	**84.96**
subseq (2,3,.25,2)	81K	185K	**83.48**	84.75
subseq (2,3,.25,1)	81K	185K	82.89	-
subseq (2,4,.5,2)	102K	206K	83.29	84.40
subseq (3,3,.5,2)	154K	259K	83.17	83.85
subseq (3,4,.25,2)	290K	-	83.06	-
subseq (3,5,.25,2)	416K	-	83.06	-
combination model				**85.40**

Figure 2.15: Accuracy of models using projection paths keyed by *le-type* or both *word* and *le-type*. Numbers of features are shown in thousands.

unifies features that have gaps in different places, and the Wildcard kernel does not. For example, $a*b, *ab, ab*$ are different features for Wildcard, but they are the same feature ab for Subsequence – only the weighting of the feature depends on the position of the wildcard.

When projection paths keyed by words are added, the accuracy increases significantly. subseq(2,3,.5,2) achieved an accuracy of 84.96%, which is much higher than the best previously published accuracy from a single model on this corpus (82.7% for a model that incorporates more sources of information from the HPSG signs (Toutanova et al., 2002)). The error reduction compared to that model is 13.1%. It is also higher than the best result from voting classifiers (84.23% (Osborne and Baldbridge, 2004)).

We should note that we performed multiple training and test runs on the same 10 folds of data. Therefore some of the differences in performance between the models cannot be trusted.

Other Features and Model Combination

Finally, we trained several models using different derivation tree annotations and built a model that combined the scores from these models together with the best model subseq(2,3,.5,2) from Figure 2.15. The combined model achieved our best accuracy of 85.4%. The models combined were:

Model I A model that uses the Node Label and le-type of non-head daughter for head projection paths, and Node Label and sysnem.local.cat.head for non-head projection paths. The model uses the subseq(2,3,.5,2) kernel for le-type keyed paths and bi-gram + Repetition for word keyed paths as above. The number of features of this model is 237K and its accuracy is 84.41%.

Model II A model that uses, for head paths, Node Label of node and Node Label and sysnem.local.cat.head of non-head daughter, and for non-head paths PS category of node. The model uses the same kernels as Model I. The number of features of this model is 311K and its accuracy is 82.75%.

Model III This model uses PS label and sysnem.local.cat.head for head paths, and only PS label for non-head paths. The kernels are the same as Model I. The number of features of this model is 165K and its accuracy is 81.91%.

Model IV This is a standard model based on rule features for local trees, with 2 levels of grandparent annotation and back-off. The annotation used at nodes is with Node Label and sysnem.local.cat.head. The number of features of this model is 78K and its accuracy is 82.6%.

2.7.4 A Note on Application to Penn Treebank Parse Trees

The representation of parse trees as leaf projection paths should be applicable to Penn Treebank parse trees, and we would expect to see the same gains from jointly modeling the head and non-head paths, and from applying string kernels to model better structural similarity.

In order to apply this representation correctly, it is important that we perform several transformations to Penn Treebank parse trees. We need to binarize the trees,

so that every dependent of a word occurs on its head path. Additionally, we need to retain information about direction (*left*, *right*) of dependent constituents with respect to a head.

One approach is to perform head-outward binarization. Thus each node in the transformed tree will represent a dependency relationship between the head and a single child. Annotation of the introduced additional nodes in the binarization with some predictive information, such as the phrase type of the dependent child, would also be important.

It would be easiest to apply this representation to Penn Treebank parsing by re-ranking top k parses from another model (Collins, 2000). Because the features are highly non-local it would be infeasible to find an exact best analysis out of millions of possible parses.

2.8 Discussion and Conclusions

We proposed a new representation of parse trees that allows us to connect more tightly tree structures to the words of the sentence. Additionally this representation allows for the natural extension of string kernels to kernels on trees. The major source of accuracy improvement for our models was this representation, as even with bi-gram features, the performance was higher than previously achieved. We were able to improve on these results by using more sophisticated Subsequence kernels and by our Repetition kernel which captures some salient properties of word projection paths.

Additionally, we proposed a Distance-weighting kernel which is a promising device for integrated modeling of symbolic and numeric properties of natural language structures.

In future work, we aim to explore the definition of new string kernels that are more suitable for this particular application and apply these ideas to Penn Treebank parse trees. We also plan to explore annotation with more features from HPSG signs.

Chapter 3

A Joint Discriminative Model for Shallow Semantic Parsing

Most work on statistical parsing has concentrated on parsing sentences into syntactic structures. However, existing syntactic annotations for sentences from broad domains, for example, newswire text in the Penn Treebank Wall Street Journal corpus (Marcus et al., 1993), do not provide enough information for extracting meaning from sentences.

In order to abstract away from syntactic variation and move toward semantic understanding of sentences, the task of *shallow semantic parsing* was defined (Fillmore, 1968; Baker et al., 1998; Palmer et al., 2005). The task is to annotate phrases in sentences with their *semantic roles* with respect to a target predicate.

In this chapter, we concentrate on learning a model for shallow semantic parsing. We assume we are given syntactic parses for sentences and want to learn a mapping from syntactic parses to semantic parses, which specify semantic role labels for argument phrases. A possible approach to this task is to learn a classifier that labels each phrase (parse tree node) independently. The main goal of our work is to look at the problem as one of learning a mapping from a syntactic structure to a joint semantic structure, which models correlations among multiple parts of the semantic structure. We study the following questions:

What kind of global information can be usefully incorporated in machine learning models for this task?

How effective is global information at improving performance?

How can such models be built so as to overcome the computational complexity of learning and search?

Defining a set of possible semantic roles for predicates is a challenging task and the subject of continuous debate (Fillmore, 1968; Jackendoff, 1972; Schank, 1972; Fillmore, 1976; Dowty, 1991). Several projects have created corpora annotated with semantic roles, for example, FrameNet (Baker et al., 1998) and the Proposition Bank (Palmer et al., 2005). We concentrate on solving this task in the context of the Proposition Bank annotations. Parts of this work were reported in (Toutanova et al., 2005a) and (Haghighi et al., 2005) and done in collaboration with Aria Haghighi.

3.1 Description of Annotations

The Proposition Bank (PropBank) labels semantic roles of verbs only. The following examples illustrate the difference between semantic annotations in PropBank and syntactic analyses. Consider the pair of sentences:

- [Harry Potter]$_{\text{AGENT}}$ *gave* [the Dursleys]$_{\text{RECIPIENT}}$ [a lesson in magic]$_{\text{THEME}}$
- [Harry Potter]$_{\text{AGENT}}$ *gave* [a lesson in magic]$_{\text{THEME}}$ [to the Dursleys]$_{\text{RECIPIENT}}$

The phrase "the Dursleys" has the same semantic role in both cases, but this is not immediately obvious from the assigned parse tree structures. Another example is the pair of sentences:

- [The wand]$_{\text{PATIENT}}$ *broke*
- [Harry Potter]$_{\text{AGENT}}$ *broke* [the wand]$_{\text{PATIENT}}$

In both sentences *the wand* is the thing broken, but in the first case it is the syntactic subject and the second it is the syntactic object. We can see that while

the Penn Treebank makes it possible to extract subject, object, and other dependent phrases, it does not indicate the roles of these phrases in the meaning of the sentence.

The PropBank corpus provides such annotations of phrases with semantic roles. The level of semantics is more shallow than for example the level of semantic representations employed in the Core Language Engine (Alshawi, 1992), but has broad coverage. PropBank contains semantic annotations for the whole Wall Street Journal part of the Penn Treebank (Marcus et al., 1993). No attempt is made to model aspect, modality, anaphora, or quantification. Nevertheless, the annotations provide a domain independent representation of meaning which abstracts away from syntax, and which should be very useful for natural language understanding tasks such as information extraction, question answering, and machine translation.

The labels for semantic roles are grouped into two groups, core argument labels and modifier argument labels, which correspond approximately to the traditional distinction between arguments and adjuncts. The meaning of the core argument labels is determined individually for each verb. The semantics of the modifier argument labels is general and consistent across verbs. The annotations are guided by specification frames for each verb v, which specify the set of allowable core arguments and their meaning. Every verb has a set of senses, and each sense can occur with a set of semantic roles, called a *roleset*. The senses were defined by expert linguists to represent different semantic and syntactic behavior of verbs. The senses are more coarse grained than WordNet senses, with an average of 1.36 per verb. Inter-annotator agreement for assigning verb senses is 94% (Palmer et al., 2005). For each sense of a verb, a set of semantic roles is specified, together with a natural language description of the meaning of each role.

The possible labels for core arguments are ARG0, ARG1, ARG2, ARG3, ARG4, ARG5, and ARGA. The semantics of core argument labels is specific to individual verbs and there are no guarantees that these argument labels have similar meanings across verbs. However, the guidelines specify that ARG0 and ARG1 are the prototypical AGENT and PATIENT respectively, for all verbs. No generalizations across verbs can be made for higher-numbered labels. In practice most statistical models built for

this task successfully use information from other verbs to assign semantic roles to the arguments of a given verb. The possible labels for modifier arguments are listed in Figure 3.1. The labels PRD and EXT can also be added to the numbered argument labels to indicate predicative uses or extent. However, in line with previous work, we ignore such uses. The labels are quite fine-grained and the inter-annotator agreement reported for assigning argument labels was 95% (Palmer et al., 2005). Here is an example specification for one sense of the verb *accept*, copied from (Palmer et al., 2005).

Frameset **accept.01** "take willingly"

- ARG0: Acceptor
- ARG1: Thing accepted
- ARG2: Accepted-from
- ARG3: Attribute
- Example: $[\text{He}]_{\text{ARG}0}$ $[\text{would}]_{\text{ARGM-MOD}}[\text{n't}]_{\text{ARGM-NEG}}$ *accept* $[\text{anything of value}]_{\text{ARG}1}$ $[\text{from those he was writing about}]_{\text{ARG}2}$. (wsj_0186)

For a thorough description of the PropBank annotations see (Palmer et al., 2005).

The annotations of PropBank are done relative to Penn Treebank parse trees. Given a target predicate v, realized by a verb and sometimes verb particles, the constituents or nodes of the syntactic parse t for a sentence s in which v occurs are annotated with semantic roles with respect to v. Each semantic role of a predicate can be filled by one or more (possibly empty) constituents. Figure 3.2 shows an example parse tree annotated with semantic roles. Often multiple nodes are labeled with the same role label in the case of split constituents or referring pronouns. Figure 3.3 shows an annotated tree that has examples of these phenomena. In the figure, the pronoun *who* is a referring pronoun that refers to the preceding phrase *the man*, and is included as part of the ARG0 argument together with *the man*. The argument ARGM-MOD is a split constituent, that is, one consisting of multiple syntactic constituents. This is because the semantic argument phrase *may or may not* is not a constituent in the parse tree. In this case the semantic argument phrase is contiguous in the

Modifier Argument Labels	
ARGM-ADV: general-purpose	ARGM-LOC: location
ARGM-EXT: extent	ARGM-DIS: discourse connective
ARGM-NEG: negation marker	ARGM-MOD: modal verb
ARGM-CAU: cause	ARGM-TMP: time
ARGM-PNC: purpose	ARGM-MNR: manner
ARGM-DIR: direction	ARGM-PRD: predication
ARGM-REC: reciprocal	

Figure 3.1: Labels of modifying arguments occurring in PropBank.

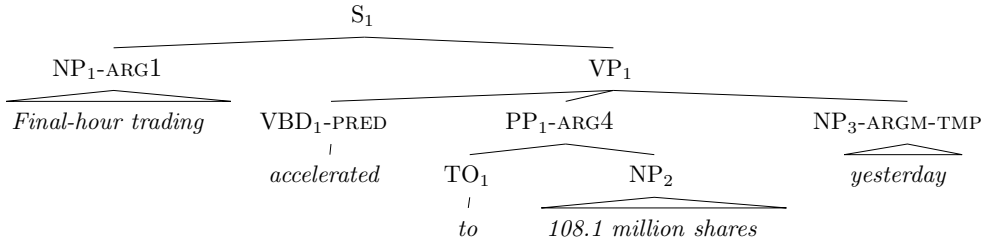


Figure 3.2: An example tree from PropBank with semantic role annotations, for the sentence *Final-hour trading accelerated to 108.1 million shares yesterday.*

sentence, but this is not always the case. The PropBank annotators were allowed to specify arguments as consisting of multiple constituents whenever this was necessary. If multiple constituents are part of the same semantic argument ARGX, we have represented this annotation by adding the label ARGX to the left-most constituent and adding the label C-ARGX to all other constituents.

3.2 Previous Approaches to Semantic Role Labeling

Traditionally, the problem of assigning semantics to sentences has been studied in a variety of computational linguistics frameworks. Hand-built precise grammars, such as Head-Driven Phrase Structure Grammar (Pollard and Sag, 1994) and Lexical Functional Grammar (Maxwell and Kaplan, 1993) must specify all mappings from syntactic realization to meaning. Such systems are usually developed for limited

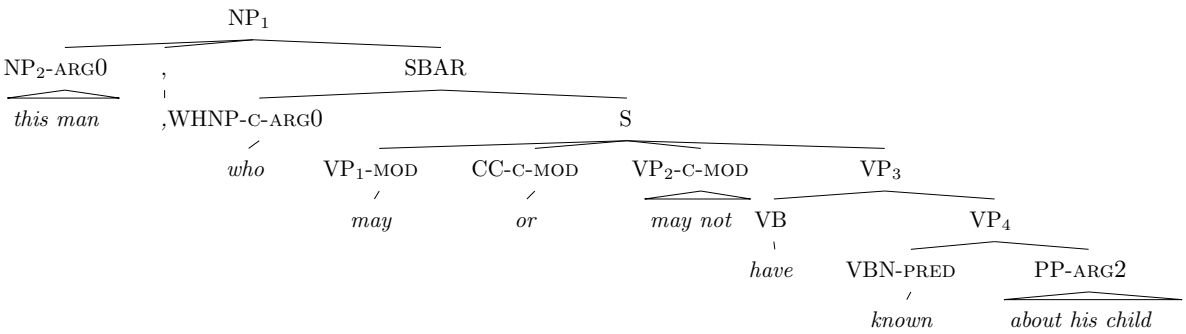


Figure 3.3: The relevant part of an example tree with semantic roles filled by multiple constituents, for the sentence *Rosie reinvented this man, who may or may not have known about his child, as a war hero for Lily's benefit.* When multiple constituents are fillers of the same semantic role ARGX, the left-most constituent is labeled ARGX, and the rest are labeled C-ARGX.

domains. Machine-learned systems for automatic assignment of semantics (Riloff, 1993; Miller et al., 1996) have been applied to slot-based semantics systems for limited domains such as the Air Traveler Information System (ATIS) spoken dialogue domain (Price, 1990), mergers and acquisitions, or terrorist attacks. Since those domains were very limited in terms of the predicates and kinds of entities used, it was possible to achieve high accuracy by shallow surface-based template matching techniques.

The first broad coverage statistical models for semantics were developed in the context of the FrameNet (Baker et al., 1998) and PropBank (Palmer et al., 2005) corpora. Even though the ontological commitments of these two corpora differ and FrameNet was developed starting with unparsed text, whereas PropBank was developed as annotations consistent with Penn Treebank parse trees, the statistical models developed in the two frameworks are easily transferrable from one to the other.

The semantic role labeling task is to assign labels to sequences of words in a sentence. The sequences of words that bear labels are usually syntactic constituents, but the current automatic syntactic parsers are not 100% percent accurate in determining constituents. Therefore it is not apriori clear what the elementary units being labeled by an automatic system should be, and whether syntactic parse information should be used. The statistical models previously developed for this task have used different units for labeling (words, syntactic chunks, constituents) and varying amounts of

syntactic information as features for predicting the labeling.

The current consensus is that the most accurate systems for PropBank use parse tree constituents as units for labeling and features from full syntactic analyses. This is shown by the results of the CoNLL shared tasks in 2004 and 2005 (Carreras and Màrquez, 2004; Carreras and Màrquez, 2005) and by several papers that specifically compared different units and different features for labeling (Gildea and Palmer, 2002; Punyakanok et al., 2005).

We therefore concentrate on using syntactic constituents as units for labeling, i.e., labeling nodes in a syntactic parse tree, and using syntactic features from the parse tree. We give an overview of previous work in this setting only. The reader is referred to, for example, (Carreras and Màrquez, 2004), for other approaches.

Before discussing the individual approaches, let us introduce some terminology. We will talk about two subtasks of the semantic role labeling task: *identification* and *classification*. In *identification*, the task is to classify nodes of the syntactic parse tree t as either ARG, an argument, or NONE, a non-argument. In *classification*, the task is, given that a node is an argument, to determine its specific semantic role label. These do not have to be sequential phases of a semantic role labeling algorithm, but it is useful to think of them as separate for two reasons. Firstly, different features may be more important for these two sub-tasks, and secondly, more efficient algorithms can be designed by exploiting this decomposition.

All current systems for semantic role labeling use as a component a *local* classifier from constituents to labels. Such a classifier uses a feature representation for constituents and assigns a label based on this feature representation. The decisions for different nodes in the parse tree are taken independently. We will use the term "*local* model" to refer to such classifiers. In terms of a probabilistic model, if the labels of all nodes are random variables, a *local* model assumes independence between any two of these variables. This independence could either be marginal or given the input features of the nodes (which exclude node labels).

It is evident that the labels and the features of arguments are highly correlated. For example, there are hard constraints – that arguments cannot overlap with each

other or the predicate, and also *soft* constraints – for example, is it unlikely that a predicate will have two or more AGENT arguments, or that a predicate used in the active voice will have a THEME argument prior to an AGENT argument. We also introduce the concept of a *joint* model for semantic role labeling. A *joint* model is one that models the dependence of node labels on labels of other nodes. We emphasize the treatment of joint information in previous work because the focus of our work is exploiting such information.

Gildea & Jurafsky 2002

The work by Gildea and Jurafsky (2002) proposed the first data driven model for broad coverage semantic role labeling in the context of the FrameNet corpus. It was subsequently applied to the PropBank corpus, with minor modifications (Gildea and Palmer, 2002; Pradhan et al., 2004). We will describe it in the context of PropBank role labeling.

The approach uses different probabilistic models for identification and classification. In identification, the estimated probability distribution is $P_{ID}(\{\text{ARG}, \text{NONE}\}|n, t, v)$ for a given node n in a syntactic tree t and a target predicate v. In classification, the estimated distribution is $P_{CLS}(r|n, t, v, \text{ARG})$, where r ranges over all possible labels of semantic roles and the distribution is conditional on the node being an argument.

The two distributions are estimated using deleted linear interpolation models over a manually specified back-off lattice, based on different features. The union of features used in the two models is presented in Figure 3.4. The features are defined relative to a node n which we are classifying and also relative to a target predicate v and the syntactic tree t. The figure also shows the values of these features for the node NP_1 in the example parse tree in Figure 3.2.

Most of the feature names displayed in the figure are self-explanatory. The syntactic path feature (PATH) shows the sequence of nodes in the parse tree on the shortest path from the constituent being classified to the predicate. Up and down arrows show direction in the parse tree. The PATH feature is indicative of the syntactic role of the constituent; for example, the example path shown in the figure is a typical subject path. The governing category GOV has only two values – *S* and *VP*, and also aims to

Feature	Value
PHRASE TYPE: Syntactic Category of node	*NP*
PREDICATE LEMMA: Stemmed Verb	*accelerate*
PATH: Path from node to predicate	*NP*↑*S*↓*VP*↓*VBD*
POSITION: Before or after predicate?	*before*
VOICE: Active or passive predicate voice	*active*
HEAD WORD OF CONSTITUENT	*trading*
SUB-CAT: CFG expansion of predicate's parent	*VP*→*VBD PP NP*
GOV: Category of first S or VP node dominating the predicate	*S*

Figure 3.4: Features used in the Gildea & Jurafsky model. The feature values shown are for the node NP_1 in Figure 3.2.

inform roughly of grammatical function. If GOV is S, the constituent is more likely to be a subject. The lexical features for the predicate and the constituent head-word aim to capture predicate-specific regularities and selectional restrictions.

All features were used in the estimation of the classification model $P_{CLS}(r|n,t,v,\text{ARG})$. The PATH feature was slightly detrimental and was excluded in some versions of the classification model. The identification model $P_{ID}(\{\text{ARG},\text{NONE}\}|n,t,v)$ used fewer features – only PATH, PREDICATE LEMMA, and HEAD WORD OF CONSTITUENT.

The final probability distribution $P_{SRL}(l|n,t,v)$ over role labels including NONE is obtained by chaining the two distributions as follows:

$$\begin{aligned} P_{SRL}(\text{NONE}|n,t,v) &= P_{ID}(\text{NONE}|n,t,v) \\ P_{SRL}(r|n,t,v) &= P_{ID}(\text{ARG}|n,t,v) \times P_{CLS}(r|n,t,v,\text{ARG}) \end{aligned}$$

Here the random variable r ranges over possible semantic role labels only, and l ranges over possible semantic role labels and the label NONE indicating a non-argument. Having learned such a local semantic role labeling model, one can select a labeling of the whole parse tree t given a target verb v by picking the best label for each node independently, or equivalently by choosing the maximizing labeling of the product of probabilities over parse tree nodes. In the work reported in (Gildea and Jurafsky, 2002), the labeling was performed in two stages – first filtering out all nodes in the parse tree for which the probability of having an argument label (ARG) according to the identification model P_{ID} is less than a threshold, and then selecting the most likely assignment to all remaining nodes according to the classification model

P_{CLS}.

The evaluation of this model was done on the FrameNet corpus, using automatic parses. The automatic parses were produced by the parser of Collins (Collins, 1997). Many of the phrases that are fillers of semantic roles do not correspond to constituents produced by the parser. However, for the FrameNet corpus, hand corrected parse trees were not available.

The performance of this model was 55.1 F-Measure if a natural threshold of .5 was used to filter constituents in identification, and 59.2 when the threshold was lowered to .35 (if the probability $P_{ID}(\text{ARG}) > .35$, the node was accepted as an argument node).

Including Joint Information

Gildea & Jurafsky propose a method to model global dependencies by including a probability distribution over multi-sets of semantic role labels given a predicate. In this way the model can look at the assignment of all nodes in the parse tree and evaluate whether the set of realized semantic roles is likely. If a necessary role is missing or if an unusual set of arguments is assigned by the local model, this additional factor can correct for some of the mistakes. The distribution over label multi-sets is estimated again using interpolation of a relative frequency and a back-off distribution. The back-off distribution assumes each argument label is present or absent independently of the other labels, i.e., it assumes a Bernoulli Naive Bayes model. The order of the roles is not modeled.

Let $L = l_1, l_2, \ldots, l_m$ denote a complete assignment of labels to nodes $n_1, n_2, \ldots, n_m$ of the parse tree t given a predicate v. Let $RoleSet(L)$ denote the assigned role multi-set according to L. $RoleSet(L)$ contains the set of non-NONE labels in L. Note that many different labelings L will correspond to the same role multi-set. Then the probability of L is given by:

$$P(L|t,v) \propto P(RoleSet(L)|v) \prod_{i=1\cdots m} \frac{P_{SRL}(l_i|n_i,v,t)}{P(l_i|v)}$$

The maximizer of this product is found approximately using re-scoring of top $k = 10$

assignments according to the local model. The division by the marginal probability $P(l_i|v)$ is included to counter the double generation of role labels.

Using this model improves the performance of the system in F-Measure from 59.2 to 62.85. This shows that adding global information improves the performance of a role labeling system considerably. However the type of global information in this model is limited to label multi-sets, and the global and local factors of the model are estimated independently and then re-normalized to counter the double generation. We will show that much larger gains are possible from joint modeling, when adding richer sources of joint information using a more principled statistical model.

Another notable model developed for FrameNet was proposed by Thompson et al in 2003 (Thompson et al., 2003). The main idea in that work was to define a joint generative model over semantic role label sequences and the parse tree t. It models the realized semantic roles as a *sequence* rather than a multi-set, and assumes a first order Markov model for the sequence. A significant gain from using a first order Markov model compared to a zero order Markov model was reported. However, the sequence of phrases to be labelled is pre-specified by heuristics applied to the parse tree to extract candidate argument phrases. Nested constituents are not considered to be candidates - the parse tree is flattened into a sequence of candidate phrases, which likely results in a significant accuracy loss. The NONE-labeled phrases are also part of the sequence.

Pradhan et al. 2004, 2005

The model proposed by Pradhan et al. in 2004 (Pradhan et al., 2004; Pradhan et al., 2005a) is a state-of-the art semantic role labeling model. The major developments over the Gildea & Jurafsky model were the use of more sophisticated local classifiers – Support Vector Machines – and the definition of new useful features. The system uses one versus all classifiers for all labels l, including the NONE label. We will not list all features used in that work. We will describe only the ones that we have used in our system in §3.5.

The system also includes joint information in two ways:

- Dynamic class context: using the labels of the two nodes to the left as features

in the one versus all classifiers. This is similar to SVM and MEMM sequence models often used in information extraction. Decoding was done using YamCha[1] (Kudo and Matsumoto, 2001) with TinySVM.[2] Notice that here the previous two nodes classified are most likely not the previous two nodes assigned non-NONE labels. If a linear order on all nodes is imposed, then the previous two nodes classified most likely bear label NONE.

- Re-scoring of an n-best lattice with a trigram language model over semantic role label sequences. The target predicate is also a part of the sequence.

These ways of incorporating joint information resulted in small gains over a baseline system using only the Gildea & Jurafsky features. The performance gain due to joint information over a system using all features was not reported.

Note that when re-scoring an n-best lattice with a language model, we end up double generating the labels. Only the top two hypotheses at each node were considered in the lattice generation, where one of the hypotheses was always NONE. Therefore, this model does not re-score different argument label options for the nodes. Global information helps only to re-rank NONE versus the most likely ARG label for each node.

The system had the best published results on PropBank in 2004 and also a similar system which did tagging of shallow parsed representations was the winner of the CoNLL 2004 shared task on semantic role labeling. Their published results in 2004 were 89.4 F-Measure on ALL arguments using gold standard parse trees and 78.8 F-Measure using automatic parses produced by Charniak's parser (Charniak, 2000). We will compare our system's results to these numbers later on, because there are differences in the scoring measures that render the raw numbers not directly comparable.

In their ACL 2005 paper (Pradhan et al., 2005b), the group introduced several improvements to the system. One of them was calibrating the SVM scores to make them behave more like probabilities. A second improvement was adding features from

[1]Available at http://chasen.org/~taku/software/yamcha/
[2]Available at http://chasen.org/~taku/software/TinySVM/

a Combinatory Categorial Grammar parse of the sentence. A third improvement was combining systems based on multiple syntactic views – constituency parsing with Charniak's parser, dependency parsing with MiniPar,[3] a chunking representation, and CCG parsing. The third improvement lead to gains only when the gold standard syntactic parse was not given.

All of these improvements resulted in performance gains. The F-Measure on gold standard parses for ALL arguments rose from 89.4 to 91.2 F-Measure and the performance on automatic parses rose from 78.8 to 81.0 (79.9 when using only the Charniak and the CCG parse).

For evaluation on gold standard parse trees, this new system is not directly comparable to previous work. This is because in previous work gold standard Penn Treebank parses have been stripped of functional tag annotations and null constituents, whereas gold standard CCG parses use such additional annotations. This is because CCG bank was created from the WSJ Penn Treebank via a mapping algorithm that uses the null constituents, traces, and functional tag annotations (for example, to detect long-distance dependencies and to distinguish arguments from adjuncts (Hockenmaier, 2003a)). In preliminary experimentation (Palmer et al., 2005) found that one can achieve significant gains by using the null constituents in the Penn Treebank (over five percentage points gain in precision and recall). The work by (Gildea and Hockenmaier, 2003) shows a similar positive results by using CCG gold standard parses for labeling, and also a two percentage point gain on core arguments when using automatic parses. We also showed that the same is true for functional tags.[4] For automatic parses it is acceptable to compare to such systems using additional annotations, as long as the null constituents and functional tags are recovered automatically.

Punyakanok et al. 2004, 2005

The main idea of the work of Punyakanok et al. (2004; Punyakanok et al. (2005) is to build a semantic role labeling system that is based on local classifiers but also uses a global component that ensures that several linguistically motivated global constraints

[3]Available at http://www.cs.ualberta.ca/~lindek/minipar.htm

[4]Unpublished experiments.

on argument frames are satisfied.

For example, one global constraint is that the argument phrases cannot overlap – i.e., if a node is labeled with a non-NONE label, all of its descendants have to be labeled NONE. Another constraint is that arguments cannot include the predicate node, but this constraint can also be handled at the local level by including a feature with weight of negative infinity for nodes that dominate the predicate.

The proposed framework is integer linear programming (ILP), which makes it possible to find the most likely assignment of labels to all nodes of the parse tree subject to our specified constraints. Solving the ILP problem is NP-hard but it is very fast in practice (Punyakanok et al., 2004).

In this framework the constraints that have been proposed to ensure global consistency are categorical – a constraint is either violated or not. The constraints themselves are not learned automatically but are specified by a knowledge engineer.

It turns out that enforcing the argument non-overlapping constraint does not lead to large gains when a single parse tree t is labeled. However, if we are allowed to label constituents in multiple parse trees, for example, the top k produced by Charniak's parser, or trees produced by Charniak's and Collins' parsers, enforcing the argument non-overlapping and the other constraints becomes much more important. Thus, by using nodes in multiple parse trees and using the ILP framework, the group from UIUC (Punyakanok et al., 2005) obtained the best performance in the CoNLL 2005 shared task.

Other Work

Several other research groups proposed new features for semantic role labeling (Xue and Palmer, 2004; Surdeanu et al., 2003; Carreras and Màrquez, 2005), and other local classifiers – based on log-linear models, decision trees, random forest, etc. A method that models joint information in a different way was also proposed by (Cohn and Blunsom, 2005). It uses a tree conditional random field, where the dependency structure is exactly defined by the edges in the syntactic parse tree. The only dependencies captured are between the label of a node and the label of each of its children. However, the arguments of the predicate can be arbitrarily far in the

syntactic parse tree and therefore a tree CRF model is quite limited.

Using Multiple Sources of Syntactic Information

In addition to building models that label the nodes in a single parse tree, several groups started looking at using multiple syntactic representations or multiple guesses of a single parser. This is expected to be helpful when none of the parsers is completely correct and each of the sources can contribute some correct information. Examples of this work include (Pradhan et al., 2005b; Punyakanok et al., 2005; Haghighi et al., 2005), and other systems that entered the CoNLL 2005 shared task (Carreras and Màrquez, 2005). Also the idea of simultaneous syntactic and semantic parsing was introduced (Yi and Palmer, 2005). These are important ideas which clearly contributed to the performance of automatic semantic role labeling systems.

In our work we concentrate on using a single gold standard or automatic parse tree and briefly report on an experiment using simple Bayesian combination over multiple guesses of a single parser.

3.3 Ideas of This Work

Linguistic intuition tells us that a core argument frame is a *joint* structure, with strong dependencies between arguments. For instance, in the sentence "[*Final-hour trading*]$_{\text{THEME}}$ *accelerated* [*to 108.1 million shares*]$_{\text{TARGET}}$ [*yesterday*]$_{\text{ARGM-TMP}}$" from Figure 3.2, the first argument is the subject noun phrase *final-hour trading* of the active verb *accelerated*. If we did not consider the rest of the sentence, it would look more like an AGENT argument, but when we realize that there is no other good candidate for a THEME argument, because *to 108.1 million shares* is a TARGET and *yesterday* is an ARGM-TMP, we would correctly label it THEME.

Even though previous work has modeled some correlations between the labels of parse tree nodes, many of the phenomena that seem very important to capture are not modeled. The key properties that would make an approach able to model this joint structure are: (*i*) no finite Markov horizon assumption for dependencies among node labels, (*ii*) features looking at the labels of multiple argument nodes and *internal*

features of these nodes, and (*iii*) a statistical model capable of incorporating these long-distance dependencies and generalizing well.

We show how to build a joint model of argument frames, incorporating novel features into a discriminative log-linear model. This system achieves an error reduction of 17% on ALL arguments and 36.8% on CORE arguments over a state-of-the art independent classifier for gold standard parse trees on PropBank.

Let us think of a graphical model over a set of m variables, one for each node in the parse tree t, representing the labels of the nodes and the dependencies between them. In order for a model over these variables to capture, for example, the statistical tendency of some semantic roles to occur at most once (e.g., there is usually at most one constituent labeled AGENT), there must be a dependency link between any two variables. To estimate the probability that a certain node gets the role AGENT, we need to know if any of the other nodes was labeled with this role.

We propose a similar model, which is globally conditioned on the observation (the parse tree) and is thus similar to a Conditional Random Field (CRF) (Lafferty et al., 2001) with a very rich graphical structure. This is very different from previous work on CRF models for sequence classification tasks in NLP, because such previous work has made strong independence assumptions (Lafferty et al., 2001; Sha and Pereira, 2003; Cohn and Blunsom, 2005), usually an order n markov assumption on the sequence of labels. For other tasks, CRF models going beyond markov independence assumptions have been proposed. Examples are work on LFG parse disambiguation (Johnson et al., 1999), Penn Treebank parse re-ranking (Collins, 2000), and our work on HPSG disambiguation from the previous chapter. In this work, we explore how one can build and do inference in a Conditional Random Field without any independence assumptions on the label sequence.

Such a rich graphical model can represent many dependencies but there are two dangers – one is that the computational complexity of training the model and searching for the most likely labeling given the tree can be prohibitive, and the other is that if too many dependencies are encoded, the model will over-fit to the training data and will not generalize well.

We propose a model which circumvents these two dangers and achieves significant performance gains over a similar local model that does not add any dependency arcs among the random variables. To tackle the efficiency problem, we adopt dynamic programming and re-ranking algorithms. To avoid overfitting we encode only a small set of linguistically motivated dependencies in features over sets of the random variables.

The re-ranking approach, similarly to the approach to parse re-ranking by (Collins, 2000), employs a simpler model – a local semantic role labeling algorithm, as a first pass to generate a set of N likely joint assignments of labels to all parse tree nodes. The joint model is restricted to these N assignments and does not have to search the exponentially large space of all other possible ones.

We start describing our system in detail by first introducing the simpler local semantic role labeling models, and later building on them to define joint models. Before we start presenting models, in §3.4 we describe the data and evaluation measures used. The reader can skip the next section and continue on to §3.5, if he/she is not interested in meticulous evaluation details.

3.4 Data and Evaluation Measures

For most of our experiments we used the February 2004 release of PropBank. We also report results on the CoNLL 2005 shared task data in §3.7.2. In this section we describe the data and evaluation measures for the February 2004 data. The CoNLL data and evaluation measures are standard and do not require thorough description.

For the February 2004 data, we used the standard split into training, development, and test sets – the annotations from sections 02–21 formed the training set, section two4 the development, and section two3 the test set. We removed seven propositions from the training section of the Feb04 data, because they contained arguments labeled ARGM-preposition, where preposition was one of *on*, *by*, *with*, *in*, *at*, and *for*. These should not be valid modifier argument labels and our model would try to learn parameter vectors for them had we not excluded these examples from the data. The set of argument labels considered is the set of core argument labels plus the modifier

labels from Figure 3.1. The training set contained 85,392 propositions, the test set 4,615, and the development set 2,626 propositions.

We evaluate semantic role labeling models on gold standard parse trees and parse trees produced by Charniak's automatic parser (Charniak, 2000), using the version from March 18, 2005. For gold standard parse trees, we preprocess the trees to discard empty constituents and strip functional tags. Using the trace information provided by empty constituents is very useful to improve performance (Palmer et al., 2005; Pradhan et al., 2005b; Gildea and Hockenmaier, 2003), but we have not used this information in order to be able to compare our results to previous work and since automatic systems that recover it are not publicly available.

There are several issues with the evaluation measures that make it hard to compare results obtained by different researchers. The first issue is the existence of arguments which consist of multiple constituents. In this case it is not clear whether partial credit is to be given for guessing only some of the constituents comprising the argument correctly.

The second issue is whether the bracketing of constituents has to be recovered correctly. In other words, the issue is whether pairs of labelings, such as "<ARG0> the </ARG0> <ARG0>man </ARG0>" and "<ARG0> the man </ARG0>", are to be considered the same or not. If they are to be considered the same, there are multiple labelings of nodes in a parse tree that are equivalent. For example, suppose there is a constituent **A** dominating words in the span $[a, b]$, where a and b are word positions. Suppose also that **A** has two children **B** and **C**, dominating spans $[a, c]$ and $[c + 1, b]$ respectively. Then the labeling of the span $[a, b]$ with a label ARG0 corresponds to labeling **A** with that label and labeling all of its descendants with NONE, or labeling **A** with NONE, **B** and **C** with ARG0, and all of their descendants with NONE.

The third issue is that when using automatic parsers, some of the constituents that are fillers of semantic roles are not recovered by the parser. In this case it is not clear how various research groups have scored their systems (using head-word match, ignoring these arguments altogether, or using exact match). We chose to require exact match rather than head-word match or other approximate matching.

Here we describe in detail our evaluation measures for the results on the February 2004 data reported in this chapter. For both gold standard and automatic parses we use two evaluation measures – constituent-based and argument-based. If we vary the choice taken for the three issues listed above, we can come up with many (at least eight) different evaluation measures. We chose to consider only two evaluation measures, both of which seem reasonable. They differ with regard to bracketing and multi-constituent arguments.

To describe the two evaluation measures, we will use as an example the correct and guessed semantic role labelings shown in Figures 3.5(a) and 3.5(b). Both are shown as labelings on parse tree nodes with labels of the form ARGX and C-ARGX. The label C-ARGX is used to represent multi-constituent arguments. A constituent labeled C-ARGX is assumed to be a continuation of the closest constituent labeled ARGX to the left. Our semantic role labeling system produces labelings of this form and the gold standard PropBank annotations are converted to this form as well.[5]

The constituent-based measures correspond more closely to the task of labeling nodes in a parse tree. They require exact bracketing and allow partial credit for labeling correctly only some of several constituents in a multi-constituent argument. The argument-based measures do not require exact bracketing and do not give partial credit for labeling correctly only some of several constituents in a multi-constituent argument. The argument-based measures are more similar to the CoNLL 2005 shared task evaluation measures. The two evaluation measures are illustrated in Figure 3.5.

More precisely, for constituent-based measures, a semantic role labeling of a sentence is viewed as a labeling on contiguous spans of words of the form $[w_i,\ldots,w_j]$-ARGX. Figure 3.5(c) gives the representation of the correct and guessed labelings shown in Figures 3.5(a) and 3.5(b), in the first and second rows of the table, respectively. The labeling $[w_i,\ldots,w_k]$-ARGX, $[w_{k+1},\ldots,w_j]$-ARGX is different from the labeling $[w_i,\ldots,w_j]$-ARGX. The labelings on parse tree nodes are converted to this

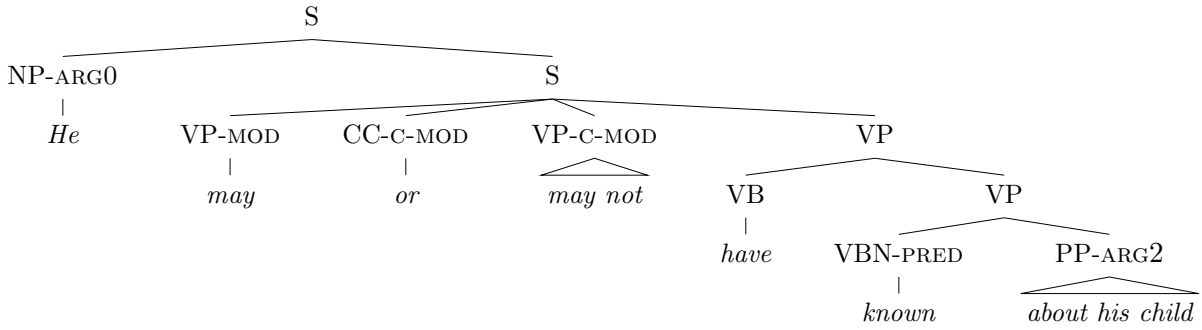
[5]This representation is not powerful enough to represent all valid labelings of multi-constituent arguments, since it cannot represent the case where a new argument with label ARGX starts before a previous multi-constituent argument with the same label ARGX has finished. However, this case should be very rare.

form by labeling each word span dominated by an argument constituent with the semantic role of the constituent, substituting C-ARGX with ARGX. All remaining contiguous word spans are implicitly labeled with NONE. The information whether several constituents that are labeled the same are part of the same argument or not is lost.

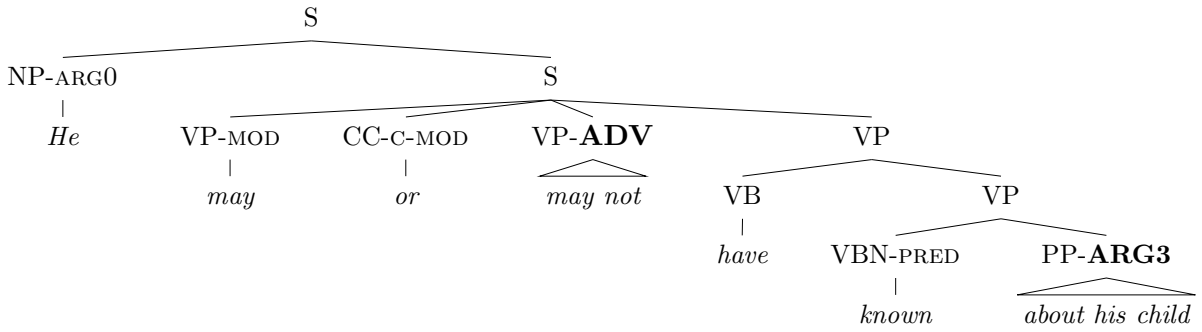
For argument-based measures, a semantic role labeling of a sentence is viewed as a labeling on sets of words. These sets can encompass several non-contiguous spans. Figure 3.5(c) gives the representation of the correct and guessed labelings shown in Figures 3.5(a) and 3.5(b), in the third and fourth rows of the table, respectively. To convert a labeling on parse tree nodes to this form, we create a labeled set for each possibly multi-constituent argument. All remaining sets of words are implicitly labeled with NONE. We can see that in this way, exact bracketing is not necessary and also no partial credit is given when only some of several constituents in a multi-constituent argument are labeled correctly.

We will refer to both contiguous word spans and word sets as "spans". For both the constituent and argument-based measures, we are comparing a guessed set of labeled spans to a correct set of labeled spans. The spans labeled NONE are omitted. We briefly define the various measures of comparison used below, using the example guessed and correct labelings shown in Figure 3.5(c). The scoring measures are illustrated in Figure 3.5(d).

The figure shows performance measures – F-Measure (F1) and Whole Frame Accuracy (Acc.) – for constituent-based and argument-based scoring across nine different conditions. When the sets of labeled spans are compared directly, we obtain the complete task measures, corresponding to the ID&CLS rows and ALL columns in Figure 3.5(d). We also define several other measures to understand the performance of the system on different types of labels. These other measures can be seen as comparing the sets of labeled spans after an appropriate transformation is first applied to them. Suppose we are comparing a correct set of labeled spans $S_{correct}$ to a guessed set of labeled spans $S_{guessed}$. To obtain the scoring measures for a row labeled ID, CLS, or ID&CLS, and a column labeled CORE, COARSEARGM, or ALL, we



(a) Correct labeling.



(b) Guessed labeling.

Type	Location	Labeling
Constituent	Correct	$[0,0]$-ARG0, $[1,1]$-MOD, $[2,2]$-MOD, $[3,4]$-MOD, $[7,9]$-ARG2
	Guessed	$[0,0]$-ARG0, $[1,1]$-MOD, $[2,2]$-MOD, $[3,4]$-ADV, $[7,9]$-ARG3
Argument	Correct	$\{0\}$-ARG0, $\{1,2,3,4\}$-MOD, $\{7,8,9\}$-ARG2
	Guessed	$\{0\}$-ARG0, $\{1,2\}$-MOD, $\{3,4\}$-ADV $\{7,8,9\}$-ARG3

(c) Labelings of spans.

Task	CORE		COARSEARGM		ALL	
	F1	Acc.	F1	Acc.	F1	Acc.
Constituent ID	100.0	100.0	100.0	100.0	100.0	100.0
Constituent CLS	50.0	0.0	80.0	0.0	60.0	0.0
Constituent ID&CLS	50.0	0.0	80.0	0.0	60.0	0.0
Argument ID	100.0	100.0	57.1	100.0	57.1	0.0
Argument CLS	50.0	0.0	50.0	0.0	50.0	0.0
Argument ID&CLS	50.0	0.0	28.6	0.0	28.6	0.0

(d) Scoring measures.

Figure 3.5: Constituent-based and argument-based scoring measures for the guessed labeling.

first apply one transformation to the two sets depending on the column, then a second transformation depending on the row, and finally compare the resulting sets of labeled spans to compute the F-Measure and Whole Frame Accuracy measures. The transformations corresponding to the column ALL and the row ID&CLS are identity transformations – i.e., we compare the original sets of labeled spans. We will describe the transformations corresponding to the other columns and rows shortly. Once we have the transformed $S_{correct}$ and $S_{guessed}$, we compute the Whole Frame Accuracy and F-Measure as follows:

Whole Frame Accuracy: (Acc.) this is the percentage of propositions for which there is an exact match between the proposed and correct labelings. For example, the whole frame accuracy under both constituent and argument-based scoring for ID&CLS and ALL is 0, because the correct and guessed sets of labeled spans shown in Figure 3.5(c) do not match exactly. Even though this measure has not been used extensively in previous work, we find it useful to track. Foremost, potential applications of role labeling may require correct labeling of all (or at least the core) arguments in a sentence in order to be effective, and partially correct labelings may not be very useful. Moreover, a joint model for semantic role labeling optimizes Whole Frame Accuracy more directly than a local model does.

F-Measure: Since there may be confusion about what we mean by F-Measure in this multi-class setting, we are defining it here. F-Measure is defined as the harmonic mean of precision and recall:

$$f = \frac{2 \times p \times r}{p + r}$$

$$p = \frac{true\ positive}{true\ positive + false\ positive}$$

$$r = \frac{true\ positive}{true\ positve + false\ negative}$$

True positive are spans with correct label one of the core or modifier argument labels (not NONE) and guessed label the same as the correct label. False positive are spans whose guessed label is non-NONE and whose correct label is different from the

guessed label. False negative are spans whose correct label is non-NONE and whose guessed label is not the same as the correct one. In the graphs we show F-Measure multiplied by 100 so that it is in the same range as Whole Frame Accuracy.

For example, for argument-based scoring, for the ID&CLS row and ALL column, we are comparing the following correct and guessed sets of labeled spans:

Correct: $\{0\}$-ARG0, $\{1,2,3,4\}$-MOD, $\{7,8,9\}$-ARG2

Guessed: $\{0\}$-ARG0, $\{1,2\}$-MOD, $\{3,4\}$-ADV $\{7,8,9\}$-ARG3

The guessed labeling has one true positive ($\{0\}$-ARG0), three false positive ($\{1,2\}$-MOD, $\{3,4\}$-ADV, and $\{7,8,9\}$-ARG3), and two false negative ($\{1,2,3,4\}$-MOD and $\{7,8,9\}$-ARG2). The F-Measure is thus .286.

For constituent-based scoring we are comparing the guessed labeling in row two to the correct labeling in row one of 3.5(c). We have three true positive, two false positive and two false negative. We thus obtain an F-Measure of .60. We can see how the argument and constituent-based measures can give a different result. Our guessed labeling has guessed two of the three constituents forming the MOD argument correctly, and constituent-based scoring gives it partial credit, but argument-based scoring does not.

Core Argument Measures: (CORE) These measures are aimed to score the system on core arguments only, without regard to modifier arguments. The transformation corresponding to the CORE column removes all spans labeled with a modifier argument from a set of labeled spans, or equivalently, labels those spans with NONE. In this way we are effectively ignoring the modifier arguments. For example, to obtain the constituent-based measures for CORE ID&CLS, we need to compare the following correct and guessed sets of labeled spans:

Correct: $[0]$-ARG0, $[7,9]$-ARG2

Guessed: $[0]$-ARG0, $[7,9]$-ARG3

The F-Measure is .50 and the Whole Frame Accuracy is 0.

Coarse Modifier Argument Measures: (COARSEARGM) Sometimes it is sufficient to know a given span has a modifier role, without knowledge of the specific role

label. In addition, deciding exact modifier argument labels was one of the decisions with highest disagreement among annotators (Palmer et al., 2005). The transformation corresponding to the CoarseARGM column substitutes all modifier labels with a generic ARGM label. Such performance measures were also used by (Xue and Palmer, 2004). We can see that for constituent-based scoring, the F-Measure for CoarseARGM is higher than for ALL , because the $[3, 4]$ span is counted as correctly labeled under CoarseARGM .

Identification Measures: (ID) This measures how well we do on the ARG vs NONE distinction. The transformation corresponding to the ID rows substitutes all non-NONE labels with a single generic ARG label. For example, to compute CORE ID we first apply the transformation for CORE which removes all non-core labeled spans, and then the transformation corresponding to ID . We obtain the following sets of labeled spans for argument-based CORE ID :

Correct: $\{0\}$-ARG, $\{7, 8, 9\}$-ARG

Guessed: $\{0\}$-ARG, $\{7, 8, 9\}$-ARG

The F-Measure is 1.0 and the Whole Frame Accuracy is 100%.

Classification Measures: (CLS) This is performance on argument spans which were also guessed to be argument spans (but possibly the exact label was wrong). In other words, these measures ignore the ARG vs. NONE confusions. They ignore all spans, which were incorrectly labeled NONE, or incorrectly labeled with an argument label, when the correct label was NONE. This is different from classification accuracy used in previous work to mean the accuracy of the system in classifying spans when the correct set of argument spans is given. The transformation corresponding to the CLS rows removes all spans from $S_{guessed}$ and $S_{correct}$, which do not occur in both sets. For example, to compute the ALL CLS measures for argument-based scoring, we need to compare the following sets of labeled spans:

Correct: $\{0\}$-ARG0, $\{7, 8, 9\}$-ARG2

Guessed: $\{0\}$-ARG0, $\{7, 8, 9\}$-ARG3

The rest of the spans were removed from both sets because they were labeled NONE

according to one of the labelings and non-NONE according to the other. The F-Measure is .50 and the Whole Frame Accuracy is 0%. Note that with our definitions the following equality holds:

ID&CLS F-Measure = ID F-Measure × CLS F-Measure

The CoNLL evaluation measure is closer to our argument-based measure, but it also distinguishes referring labels of the form R-ARGX. These labels are assigned to constituents which are a part of multi-constituent arguments, and which represent referring pronouns. The decision of which constituents were to be labeled with referring labels was made by using a set of rules, which are not publicly available. Therefore, it is not clear whether information in addition to PropBank annotations is necessary for these decisions.

Most of the results we report in this chapter use the argument-based measures, but we also mention some key constituent-based results for closer comparison to previous work. We also report results on the CoNLL 2005 shared task in §3.7.2.

3.5 Local Classifiers

As defined earlier, a classifier is local if it assigns a probability (or score) to the label of an individual parse tree node n_i independently of the labels of other nodes.

We use the standard separation of the task of semantic role labeling into *identification* and *classification* phases. Formally, let L denote a mapping of the nodes in t to a label set of semantic roles (including NONE) and let $Id(L)$ be the mapping which collapses L's non-NONE values into ARG. Then, like in the Gildea & Jurafsky system, we can decompose the probability of a labeling L into probabilities according to an identification model P_{ID} and a classification model P_{CLS}.

$$\begin{aligned} P_{SRL}(L|t,v) &= P_{ID}(Id(L)|t,v) \times \\ & \quad P_{CLS}(L|t,v,Id(L)) \end{aligned} \tag{3.1}$$

This decomposition does not encode any independence assumptions, but is a useful

way of thinking about the problem. Our local models for semantic role labeling use this decomposition. We use the same features for local identification and classification models, but use the decomposition for efficiency of training. The identification models are trained to classify each node in a parse tree as ARG or NONE, and the classification models are trained to label each argument node in the training set with its specific label. In this way the training set for the classification models is smaller. Note that we do not do any hard pruning at the identification stage in testing and can find the exact labeling of the complete parse tree, which is the maximizer of Equation 3.1. Thus we do not have an accuracy loss as in the two-pass hard prune strategy described in (Pradhan et al., 2005a) and we do not have to fit a threshold for the identification model as in the Gildea & Jurafsky or the Xue & Palmer (Xue and Palmer, 2004) system.

We use log-linear models for multi-class classification for the local models. One advantage of log-linear models over SVMs for us is that they produce probability distributions and thus identification and classification models that can be chained in a principled way, as in Equation 3.1.

The baseline features we used for local identification and classification models are outlined in Figure 3.6. These features are a subset of features used in previous work. The standard features at the top of the figure were defined by (Gildea and Jurafsky, 2002), and the rest are other useful lexical and structural features identified in more recent work (Pradhan et al., 2004; Surdeanu et al., 2003; Xue and Palmer, 2004). We also incorporated several novel features, described in the next subsection.

3.5.1 Additional Features for Displaced Constituents

We found that a large source of errors for ARG0 and ARG1 stemmed from cases such as those illustrated in Figure 3.7, where arguments were dislocated by raising or control verbs. Here, the predicate, *expected*, does not have a subject in the typical position – indicated by the empty NP – since the auxiliary *is* has raised the subject to its current position. In order to capture this class of examples, we use a binary feature, MISSING SUBJECT, indicating whether the predicate is "missing" its subject, and use

Standard Features (Gildea and Jurafsky, 2002)
Phrase Type: Syntactic Category of node
Predicate Lemma: Stemmed Verb
Path: Path from node to predicate
Position: Before or after predicate?
Voice: Active or passive relative to predicate
Head Word of Phrase
Sub-Cat: CFG expansion of predicate's parent
Additional Features (Pradhan et al., 2004; Surdeanu et al., 2003)
First/Last Word
Left/Right Sister Phrase-Type
Left/Right Sister Head Word/POS
Parent Phrase-Type
Parent POS/Head-Word
Ordinal Tree Distance: Phrase Type concatenated with length of Path feature
Node-LCA Partial Path: Path from constituent to lowest common ancestor with predicate node
PP Parent Head Word: If parent is a PP, parent's head word
PP NP Head Word/POS: For a PP, the head Word / POS of its rightmost NP
Selected Pairs (Xue and Palmer, 2004)
Predicate Lemma & Path
Predicate Lemma & Head Word
Predicate Lemma & Phrase Type
Voice & Position
Predicate Lemma & PP Parent Head Word

Figure 3.6: Baseline Features

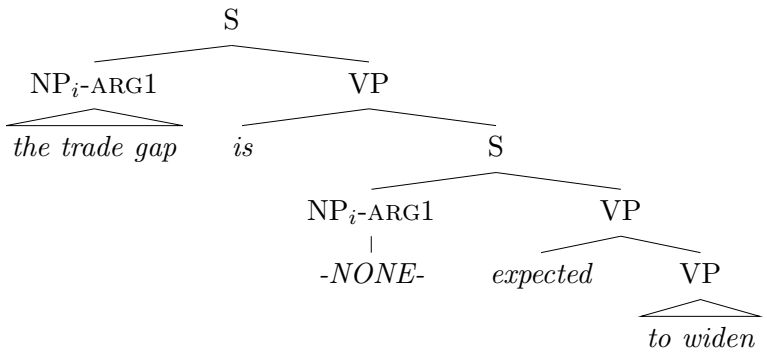


Figure 3.7: Example of displaced arguments

this feature in conjunction with the PATH feature, so that we learn typical paths to raised subjects conditioned on the absence of the subject in its typical position.

In the particular case of Figure 3.7, there is another instance of an argument being quite far from its predicate. The predicate *widen* shares the phrase *the trade gap* with *expect* as a ARG1 argument. However, as *expect* is a raising verb, *widen*'s subject is not in its typical position either, and we should expect to find it in the same positions as *expected*'s subject. This indicates it may be useful to use the path relative to *expected* to find arguments for *widen*. In general, to identify certain arguments of predicates embedded in auxiliary and infinitival VPs we expect it to be helpful to take the path from the maximum extended projection of the predicate – the highest VP in the chain of VP's dominating the predicate. We introduce a new path feature, PROJECTED PATH, which takes the path from the maximal extended projection to an argument node. This feature applies only when the argument is not dominated by the maximal projection, (e.g., direct objects). These features also handle other cases of discontinuous and non-local dependencies, such as those arising due to control verbs. The performance gain from these new features was notable, especially in identification. The performance on ALL arguments (argument-based scoring) for the model using only the features in Figure 3.6 and the model using the additional features as well, are shown in Figure 3.8.

Task	Features in Figure 3.6		+ Additional Features	
	F1	Acc.	F1	Acc.
ID	91.7	77.6	92.4	79.0
CLS	95.6	90.6	95.7	90.8
ID&CLS	87.6	70.8	88.4	72.3

Figure 3.8: Performance of local classifiers on ALL arguments, using the features in Figure 3.6 only and using the additional local features. Argument-based scoring using gold standard parse trees on section 23.

3.5.2 Enforcing the Non-overlapping Constraint

The most direct way to use trained local identification and classification models in testing is to select a labeling L of the parse tree that maximizes the product of the probabilities according to the two models as in Equation 3.1. Since these models are local, this is equivalent to independently maximizing the product of the probabilities of the two models for the label l_i of each parse tree node n_i as shown below in Equation 3.2.

$$\begin{aligned} P^{\ell}_{SRL}(L|t,v) &= \prod_{n_i \in t} P_{ID}(Id(l_i)|t,v) \\ &\times \prod_{n_i \in t} P_{CLS}(l_i|t,v,Id(l_i)) \end{aligned} \tag{3.2}$$

A problem with this approach is that a maximizing labeling of the nodes could possibly violate the constraint that argument nodes should not overlap with each other. Therefore, to produce a consistent set of arguments with local classifiers, we must have a way of enforcing the non-overlapping constraint.

Previous work has either used greedy algorithms to find a non-overlapping assignment, or the expensive ILP approach of (Punyakanok et al., 2004). Here we describe a fast exact dynamic programming algorithm to find the most likely non-overlapping (consistent) labeling of all nodes in the parse tree, according to a product of probabilities from local models, as in Equation 3.2. For simplicity, we describe the dynamic program for the case where only two classes are possible – ARG and NONE. The

generalization to more classes is straightforward. Intuitively, the algorithm is similar to the Viterbi algorithm for context-free grammars, because we can describe the non-overlapping constraint by a "grammar" that disallows ARG nodes to have ARG descendants.

Below we will talk about maximizing the sum of the logs of local probabilities rather than the product of local probabilities, which is equivalent. The dynamic program works from the leaves of the tree up and finds a best assignment for each tree, using already computed assignments for its children. Suppose we want the most likely consistent assignment for subtree t with children trees $t_1, \ldots, t_k$ each storing the most likely consistent assignment of nodes it dominates as well as the log-probability of the assignment of all nodes it dominates to NONE. The most likely assignment for t is the one that corresponds to the maximum of:

- The sum of the log-probabilities of the most likely assignments of the children subtrees $t_1, \ldots, t_k$ plus the log-probability for assigning the node t to NONE
- The sum of the log-probabilities for assigning all of t_i's nodes to NONE plus the log-probability for assigning the node t to ARG.

Propagating this procedure from the leaves to the root of t, we have our most likely non-overlapping assignment. By slightly modifying this procedure, we obtain the most likely assignment according to a product of local identification and classification models. We use the local models in conjunction with this search procedure to select a most likely labeling in testing.

It turns out that enforcing the non-overlapping constraint does not lead to large gains in performance. The results in Figure 3.8 are from models that use the dynamic program for selecting non-overlapping arguments. To evaluate the gain from enforcing the constraint, Figure 3.9 shows the performance of the same local model using all features, when the dynamic program is used versus when a most likely possibly overlapping assignment is chosen in testing.

The local model with basic + additional features with resolving overlaps is our first pass model used in re-ranking to learn a joint model. This is a state of the art model.

Enforcing constraint	CORE		ALL	
	F1	Acc.	F1	Acc.
No	90.3	81.2	88.3	71.8
Yes	90.5	81.4	88.4	72.3

Figure 3.9: Performance of local model on ALL arguments when enforcing the non-overlapping constraint or not.

Its F-Measure on ALL arguments is 88.4 with argument-based scoring and 89.9 with constituent-based scoring. This is very similar to the best previously reported results using gold standard parse trees without null constituents and functional tags – 89.4 F-Measure reported for the Pradhan et al (Pradhan et al., 2004) model. The results in (Pradhan et al., 2004) are based on a measure which is more similar to constituent-based scoring than argument-based scoring but does not correspond exactly to either of them.[6] The differences are due to the many possible choice points in defining the scoring measures. Therefore, at present, we can compare our results to those of other research only on the standard CoNLL evaluation measures.

A more detailed analysis of the results obtained by the local model is given in Figure 3.10(a), and the two confusion matrices in Figures 3.10(b) and 3.10(c), which display the number of errors of each type that the model made. The first confusion matrix concentrates on CORE arguments and merges all modifying argument labels into a single ARGM label. The second confusion matrix concentrates on confusions among modifying arguments.

From the confusion matrix in Figure 3.10(b), we can see that the largest number of errors are confusions of argument labels with NONE. The number of confusions between pairs of core argument is low, as is the number between core and modifier labels. If we ignore the column and row corresponding to NONE in Figure 3.10(b), the number of off-diagonal entries is very small. This corresponds to the high F-Measures on CoarseARGM Cls and CORE Cls, 98.1 and 98.0, respectively, shown in Figure 3.10(a). The number of confusions of argument labels with NONE, shown in the NONE *column* are larger than the number of confusions of non-NONE with argument labels, shown in the NONE *row*. This shows that the model generally has higher

[6]Personal communication, July, 2005.

precision than recall. We experimented with the precision-recall tradeoff but this did not result in an increase in F-Measure.

From the confusion matrix in Figure 3.10(c) we can see that the number of confusions between modifier argument labels is higher than the confusions between core argument labels. This corresponds to the ALL CLS F-Measure of 95.7 versus the CORE CLS F-Measure of 98.0. The per-label F-Measures, shown in the last column show that the performance on some very frequent modifier labels is in the low sixties or seventies. The confusions between modifier labels and NONE are quite numerous.

In general, to improve the performance on CORE arguments, we need to improve the recall without lowering precision, i.e., when the model is uncertain which of several likely CORE labels to assign, find additional sources of evidence to improve the confidence. To improve the performance on modifier arguments, we need to also lower the confusions among different modifier arguments. We will see that our joint model improves the overall performance mainly by improving the performance on CORE arguments, through increasing recall and precision by looking at wider sentence context.

3.5.3 On Split Constituents

As discussed is §3.4, multiple constituents can be part of the same semantic argument as specified by PropBank. An automatic system that has to recover such information needs to have a way of indicating when multiple constituents labeled with the same semantic role are a part of the same argument. Some researchers (Pradhan et al., 2004; Punyakanok et al., 2004) have chosen to make labels of the form C-ARGX distinct argument labels and that become additional classes in a multi-class constituent classifier. These C-ARGX are used to indicate continuing arguments as illustrated in the two trees in Figure 3.5. We chose to not introduce additional labels of this form, because they might unnecessarily fragment the training data. Our automatic classifiers label constituents with one of the core or modifier semantic role labels, and a simple post-processing rule is applied to the output of the system to determine

Task	CORE		CoarseARGM		ALL	
	F1	Acc.	F1	Acc.	F1	Acc.
Id	92.3	83.7	92.4	79.0	92.4	79.0
Cls	98.0	96.8	98.1	95.9	95.7	90.8
Id&Cls	**90.5**	**81.4**	90.6	76.2	**88.4**	**72.3**

(a) Summary performance results.

	Guessed								
Correct	ARG0	ARG1	ARG2	ARG3	ARG4	ARG5	ARGM	NONE	F-Measure
ARG0	2912	22	1	0	0	0	4	248	91.7
ARG1	69	3964	15	1	1	0	12	302	91.8
ARG2	7	25	740	3	2	0	9	151	82.4
ARG3	1	5	3	83	1	0	5	36	70.3
ARG4	0	1	3	0	63	0	0	7	88.1
ARG5	0	0	0	0	0	5	0	0	100.0
ARGM	0	7	10	0	0	0	2907	322	91.0
NONE	173	248	87	15	2	0	204	–	–

(b) CoarseARGM confusion matrix.

	Guessed															
Correct	ADV	CAU	DIR	DIS	EXT	LOC	MNR	MOD	NEG	PNC	PRD	REC	TMP	CORE	NONE	F-Measure
ADV	295	3	0	13	3	10	35	0	0	5	0	0	20	1	51	71.3
CAU	0	48	0	1	0	2	3	0	0	2	0	0	3	0	6	81.4
DIR	0	0	40	0	0	0	6	0	0	0	0	0	1	2	25	61.1
DIS	13	0	0	214	0	3	2	0	0	0	0	0	8	0	31	79.9
EXT	2	0	0	1	17	0	5	0	0	0	0	0	0	2	5	63.0
LOC	4	0	0	2	0	251	3	0	0	2	1	0	8	1	45	77.5
MNR	17	0	5	0	2	12	196	0	0	0	0	0	4	5	66	65.8
MOD	0	0	0	0	0	0	0	453	0	0	0	0	0	0	2	99.4
NEG	0	0	0	0	0	0	0	0	200	0	0	0	0	0	2	99.0
PNC	4	2	0	0	0	1	0	0	0	59	0	0	5	3	26	64.8
PRD	1	0	1	0	0	0	0	0	0	1	1	0	0	1	0	28.6
REC	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0.0
TMP	23	0	0	4	0	11	3	0	1	1	0	0	874	2	61	88.7
CORE	4	0	2	2	0	0	6	0	0	7	0	0	9	7927	744	92.3
NONE	28	0	9	28	0	41	30	3	1	5	0	0	59	525	–	–

(c) Modifier arguments confusion matrix.

Figure 3.10: Performance measures for local model using all local features and enforcing the non-overlapping constraint. Results are on Section 23 using gold standard parse trees and argument-based scoring.

which constituents that are labeled the same are to be merged as the same argument. This decision does not matter for constituent-based scoring, but is important for argument-based scoring. The post-processing rule is the following: for every constituent that bears a core argument label ARGX, if there is a preceding constituent with the same label, re-label the current constituent C-ARGX. Therefore, according to our algorithm, all constituents having the same core argument label are part of the same argument, and all constituents having the same modifier labels are separate arguments by themselves. This rule is very accurate for core arguments, but it sometimes fails on modifier arguments. An evaluation of this rule on the CoNLL evaluation measure shows that our upper bound in performance because of this rule is about 99.0 F-Measure on ALL arguments.

3.6 Joint Classifiers

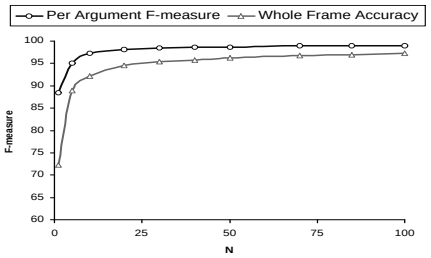
We proceed to describe our models incorporating dependencies between labels of nodes in the parse tree. As we discussed briefly before, the dependencies we would like to model are highly non-local. A factorized sequence model that assumes a finite Markov horizon, such as a chain Conditional Random Field (Lafferty et al., 2001), would not be able to encode such dependencies. We define a Conditional Random Field but with a much richer dependency structure.

The Need for Re-ranking

For argument identification, the number of possible assignments for a parse tree with n nodes is 2^n. This number can run into the hundreds of billions for a normal-sized tree. For argument labeling, the number of possible assignments is $\approx 20^m$, if m is the number of arguments of a verb (typically between 2 and 5), and 20 is the approximate number of possible labels if considering both core and modifying arguments. Training a model which has such a huge number of classes is infeasible if the model does not factorize due to strong independence assumptions. Therefore, in order to be able to incorporate long-range dependencies in our models, we chose to adopt a re-ranking

N	CORE		ALL	
	F1	Acc.	F1	Acc.
1	90.5	81.4	88.4	72.3
5	96.9	93.5	95.0	88.9
10	98.0	95.6	97.3	92.1
30	98.9	97.5	98.4	95.4
100	99.4	98.8	99.0	97.2

(a) Tabular form.



(b) Graphical form.

Figure 3.11: Oracle upper bounds for top N non-overlapping assignments from local model on CORE and ALL arguments. Using gold standard parse trees and argument-based scoring.

approach (Collins, 2000), which selects from likely assignments generated by a model which makes stronger independence assumptions. We utilize the top N assignments of our local semantic role labeling model P^{ℓ}_{SRL} to generate likely assignments. As can be seen from Figure 3.11(a), for relatively small values of N, our re-ranking approach does not present a serious bottleneck to performance. We used a value of $N = 10$ for training. In Figure 3.11(a) we can see that if we could pick, using an oracle, the best assignment out of the top 10 assignments according to the local model, we would achieve an F-Measure of 97.3 on all arguments. Increasing the number of N to 30 results in a very small gain in the upper bound on performance and a large increase in memory requirements. We therefore selected $N = 10$ as a good compromise.

Generation of top N most likely joint assignments

We generate the top N most likely non-overlapping joint assignments of labels to nodes in a parse tree according to a local model P^{ℓ}_{SRL}, by an exact dynamic programming algorithm, which is a generalization of the algorithm for finding the top non-overlapping assignment described in section 3.5.2.

Parametric Models

We learn log-linear re-ranking models for joint semantic role labeling, which use feature maps from a parse tree and label sequence to a vector space. The form of the models is as follows. Let $\Phi(t, v, L) \in \mathbb{R}^s$ denote a feature map from a tree t, target verb v, and joint assignment L of the nodes of the tree, to the vector space $\mathbb{R}^s$. Let $L_1, L_2, \cdots, L_N$ denote top N possible joint assignments. We learn a log-linear model with a parameter vector W, with one weight for each of the s dimensions of the feature vector. The probability (or score) of an assignment L according to this re-ranking model is defined as:

$$P^r_{SRL}(L|t, v) = \frac{e^{<\Phi(t,v,L),W>}}{\sum_{j=1}^{N} e^{<\Phi(t,v,L_j),W>}} \tag{3.3}$$

The score of an assignment L not in the top N is zero. We train the model to maximize the sum of log-likelihoods of the best assignments minus a quadratic regularization term.

In this framework, we can define arbitrary features of labeled trees that capture general properties of predicate-argument structure.

Joint Model Features

We will introduce the features of the joint re-ranking model in the context of the example parse tree shown in Figure 3.2, which is repeated for convenience in Figure 3.12. We model dependencies not only between the label of a node and the labels of other nodes, but also dependencies between the label of a node and input features of other argument nodes. The features are specified by instantiation of templates and the value of a feature is the number of times a particular pattern occurs in the labeled tree.

Templates

For a tree t, predicate v, and joint assignment L of labels to the nodes of the tree, we define the *candidate argument sequence* as the sequence of non-NONE labeled nodes

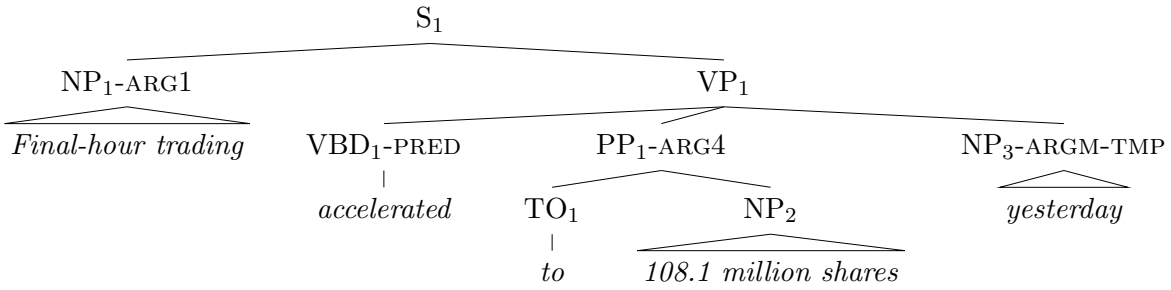


Figure 3.12: An example tree from PropBank with semantic role annotations, for the sentence *Final-hour trading accelerated to 108.1 million shares yesterday.*

$[n_1, l_1, \ldots, v_{PRED}, \ldots, n_m, l_m]$ (l_i is the label of node n_i). A reasonable candidate argument sequence usually contains very few of the nodes in the tree – about 2 to 7 nodes, as this is the typical number of arguments for a verb. To make it more convenient to express our feature templates, we include the predicate node v in the sequence. This sequence of labeled nodes is defined with respect to the left-to-right order of constituents in the parse tree. Since non-NONE labeled nodes do not overlap, there is a strict left-to-right order among these nodes. The candidate argument sequence that corresponds to the correct assignment in Figure 3.2 will be:

[NP_1-ARG1, VBD_1-PRED, PP_1-ARG4, NP_3-ARGM-TMP]

1. **Features from Local Models:** All features included in the local models are also included in our joint models. In particular, each template for local features is included as a joint template that concatenates the local template and the node label. For example, for the local feature PATH, we define a joint feature template, that extracts PATH from every node in the candidate argument sequence and concatenates it with the label of the node. Both a feature with the specific argument label is created and a feature with the generic back-off ARG label. This is similar to adding features from identification and classification models. In the case of the example candidate argument sequence above, for the node NP_1 we have the features:

   {(NP↑S↓VP↓VBD)-ARG1, (NP↑S↓VP↓VBD)-ARG}

   When comparing a local and a joint model, we use the same set of local feature

templates in the two models. If these were the only features that a joint model used, we would expect its performance to be roughly the same as the performance of a local model. This is because the two models will in fact be in the same parametric family but will only differ slightly in the way the parameters are estimated. In particular, the likelihood of an assignment according to the joint model with local features will differ from the likelihood of the same assignment according to the local model only in the denominator (the partition function Z). The joint model sums over a few likely assignments in the denominator, whereas the local model sums over all assignments; also, the joint model does not treat the decomposition into identification and classification models in exactly the same way as the local model.

2. **Whole Label Sequence:** As observed in previous work (Gildea and Jurafsky, 2002; Pradhan et al., 2004), including information about the set or sequence of labels assigned to argument nodes should be very helpful for disambiguation. For example, including such information will make the model less likely to pick multiple nodes to fill the same role or to come up with a labeling that does not contain an obligatory argument. We added a whole label sequence feature template that extracts the labels of all argument nodes, and preserves information about the position of the predicate. The template also includes information about the voice of the predicate. For example, this template will be instantiated as follows for the example candidate argument sequence:

   [voice:active, ARG1, PRED, ARG4, ARGM-TMP]

   We also add a variant of this feature which uses a generic ARG label instead of specific labels. This feature template has the effect of counting the number of arguments to the left and right of the predicate, which provides useful global information about argument structure. As previously observed (Pradhan et al., 2004), including modifying arguments in sequence features is not helpful. This corresponds to the standard linguistic understanding of adjuncts and was confirmed in our experiments. We redefined the whole label sequence features to exclude modifying arguments.

One important variation of this feature uses the actual predicate lemma in addition to "voice:active".

3. **Joint Syntactic-Semantic Features:** this class of features is similar to the whole label sequence features, but in addition to labels of argument nodes, it includes syntactic features of the nodes. These features can capture the joint mapping from the syntactic realization of the predicates's arguments to its semantic frame. The idea of these features is to capture knowledge about the label of a constituent given the syntactic realization and labels of all other arguments of the verb. This is helpful to capture syntactic alternations, such as the dative alternation. For example, consider the sentence (*i*) "[Shaw Publishing]$_{\text{ARG}_0}$ *offered* [Mr. Smith]$_{\text{ARG2}}$ [a reimbursement]$_{\text{ARG}_1}$" and the alternative realization (*ii*) "[Shaw Publishing]$_{\text{ARG}_0}$ *offered* [a reimbursement]$_{\text{ARG}_1}$ [to Mr. Smith]$_{\text{ARG}_2}$". When classifying the NP in object position, it is useful to know whether the following argument is a PP. If yes, the NP will more likely be an ARG_1, and if not, it will more likely be an ARG_2. A feature template that captures such information extracts, for each argument node, its phrase type and label. For example, the instantiation of such a template in (*ii*) would be

   [voice:active, NP-ARG0, PRED, NP-ARG1, PP-ARG2]

   We also add a template that concatenates the identity of the predicate lemma itself. We experimented with variations for what syntactic information to extract from each argument node and found that the phrase type and the head of a directly dominating PP – if one exists – was most helpful.

   We should note that Xue and Palmer (2004) define a similar feature template, called *syntactic frame*, which often captures similar information. The important difference is that their template extracts contextual information from noun phrases surrounding the predicate, rather than from the sequence of argument nodes. Because we use a joint model, we are able to use information about other argument nodes when labeling a node.

4. **Repetition Features**

   We also add features that detect repetitions of the same label in a candidate

argument sequence, together with the phrase types of the nodes labeled with that label. For example, (NP-ARG0, WHNP-ARG0) is a common pattern of this form. A variant of this feature template also indicates whether all repeated arguments are sisters in the parse tree, or whether all repeated arguments are adjacent in terms of word spans. These features can provide robustness to parser errors, making it more likely to label adjacent phrases incorrectly split by the parser with the same label.

We report results from the joint model and an ablation study of the contribution of each of the types of joint features in §3.6.1.

Final Pipeline

Here we describe the application in testing of a joint model for semantic role labeling, using a local model P^{ℓ}_{SRL}, and a joint re-ranking model P^{r}_{SRL}. P^{ℓ}_{SRL} is used to generate top N non-overlapping joint assignments $L_1, \ldots, L_N$.

One option is to select the best L_i according to P^{r}_{SRL}, as in Equation 3.3, ignoring the score from the local model. In our experiments, we noticed that for larger values of N, the performance of our re-ranking model P^{r}_{SRL} decreased. This was probably due to the fact that at test time the local classifier produces very poor argument frames near the bottom of the top N for large N. Since the re-ranking model is trained on relatively few good argument frames, it cannot easily rule out very bad frames. It makes sense then to incorporate the local model into our final score. Our final score is given by:

$$P_{SRL}(L|t,v) = (P^{\ell}_{SRL}(L|t,v))^{\alpha}\ P^{r}_{SRL}(L|t,v)$$

where α is a tunable parameter for how much influence the local score has in the final score.[7] Such interpolation with a score from a first-pass model was also used for parse re-ranking in (Collins, 2000). Given this score, at test time we choose among

[7] We found $\alpha = 1.0$ to work best.

the top N local assignments $L_1, \ldots, L_N$ according to:

$$\text{argmax}_{L \in L_1, \ldots, L_N} \; \alpha \log P^{\ell}_{SRL}(L|t, v) + \log P^{r}_{SRL}(L|t, v) \tag{3.4}$$

3.6.1 Joint Model Results

We compare the performance of joint re-ranking models and local models. We used N=10 joint assignments for training re-ranking models, and N=15 for testing. The weight α of the local model was set to 1.

Figure 3.13 shows the summary performance of the local model (LOCAL, repeated from earlier figures), a joint model using only local features (JOINTLOCAL), a joint model using local + whole label sequence features (LABELSEQ), and a joint model using all described types of features (ALLJOINT). The evaluation is on gold standard parse trees, using argument-based scoring. In addition to performance measures, the figure shows the number of binary features included in the model. The number of features is a measure of the complexity of the hypothesis space of the parametric model.

We can see that a joint model using only local features outperforms a local model by .5 points of F-Measure. The joint model using local features estimates the feature weights only using the top N consistent assignments thus making the labels of different nodes non-independent according to the estimation procedure, which may be a cause for the improved performance. Another factor could be that the model JOINTLOCAL is a combination of two models as specified in Equation 3.4, which may lead to gains as is usual for classifier combination.

The label sequence features added in Model LABELSEQ result in another .5 points jump in F-Measure on all arguments. An additional larger .8 gain results from the inclusion of syntactic-semantic and repetition features. The error reduction of model ALLJOINT over the local model is 36.8% in CORE arguments F-Measure, 33.3% in CORE arguments whole frame accuracy, 17.0% in ALL arguments F-Measure, and 21.7% in ALL arguments whole frame accuracy.

All differences in ALL arguments F-Measure are statistically significant according

Model	Num Features	CORE		CoarseARGM		ALL	
		F1	Acc.	F1	Acc.	F1	Acc.
Local	5,201K	90.5	81.4	90.6	76.2	88.4	72.3
JointLocal	2,193K	90.9	82.6	91.1	78.3	88.9	74.3
LabelSeq	2,357K	92.9	86.1	92.6	81.4	90.4	77.0
AllJoint	2,811K	94.0	87.6	93.4	82.7	**91.2**	**78.3**

Figure 3.13: Performance of local and joint models on Id&Cls on section 23, using gold standard parse trees. The number of features of each model is shown in thousands.

to a paired Wilcoxon signed rank test. JointLocal is significantly better than Local ($p < 2 \times 10^{-6}$), LabelSeq is significantly better than JointLocal ($p < 2.2 \times 10^{-16}$), and AllJoint is significantly better than LabelSeq ($p < 6 \times 10^{-8}$). We performed the Wilcoxon signed rank test on per-proposition ALL arguments F-Measure for all models.

We can also note that the joint models have less features than the local model. This is due to the fact that the local model has seen many more negative examples and therefore more unique features. The joint features are not very numerous compared to the local features in the joint models. The AllJoint model has around 30% more features than the JointLocal model.

A more detailed analysis of the results obtained by the joint model AllJoint is given in Figure 3.14(a) (Summary results), and the two confusion matrices in Figures 3.14(b) and 3.14(c), which display the number of errors of each type that the model made. The first confusion matrix concentrates on CORE arguments and merges all modifying argument labels into a single ARGM label. The second confusion matrix concentrates on confusions among modifying arguments. This Figure can be compared to Figure 3.5.2, which summarizes the results for the local model in the same form. The biggest differences are in the performance on CORE arguments, which can be seen by comparing the confusion matrices in Figures 3.10(b) and 3.14(b). The F-Measure on each of the core argument labels has increased by at least three points; the F-Measure on ARG2 has increased by 5.7 points, and the F-Measure on ARG3 by eight points. The confusions of core argument labels with NONE have gone down

Task	CORE		CoarseARGM		ALL	
	F1	Acc.	F1	Acc.	F1	Acc.
Id	95.6	89.2	95.0	85.0	95.0	85.0
Cls	98.3	97.6	98.3	96.6	96.0	91.4
Id&Cls	94.0	87.6	93.4	82.7	91.2	78.3

(a) Summary performance results.

	Guessed								
Correct	ARG0	ARG1	ARG2	ARG3	ARG4	ARG5	ARGM	NONE	F-Measure
ARG0	3025	23	3	0	0	0	5	131	94.8
ARG1	39	4147	17	1	0	0	7	153	95.0
ARG2	6	34	821	2	2	0	9	63	88.1
ARG3	0	7	2	99	0	0	5	21	78.3
ARG4	0	0	2	0	68	0	0	4	93.8
ARG5	0	0	0	0	0	5	0	0	100.0
ARGM	0	14	11	1	0	0	2975	245	92.0
NONE	124	137	70	16	1	0	217	–	–

(b) CoarseARGM confusion matrix.

	Guessed															
Correct	ADV	CAU	DIR	DIS	EXT	LOC	MNR	MOD	NEG	PNC	PRD	REC	TMP	CORE	NONE	F-Measure
ADV	307	2	0	13	3	11	39	0	0	4	0	0	20	3	34	72.9
CAU	2	47	0	1	0	2	3	0	0	2	0	0	3	0	5	80.3
DIR	0	0	44	0	0	0	6	0	0	0	0	0	0	4	20	64.2
DIS	12	0	0	217	0	2	2	0	0	0	0	0	7	0	31	81.0
EXT	2	0	0	1	16	0	3	0	0	0	0	0	0	4	6	61.5
LOC	4	0	1	3	0	250	4	0	0	1	1	0	10	2	41	76.8
MNR	17	0	5	0	1	13	224	0	0	1	0	0	6	4	36	69.7
MOD	0	0	0	0	0	0	0	453	0	0	0	0	0	0	2	99.4
NEG	0	0	0	0	0	0	0	0	199	0	0	0	0	0	3	98.8
PNC	3	2	1	0	0	0	0	0	0	74	0	0	3	4	13	76.3
PRD	1	0	1	0	0	0	0	0	0	0	2	0	0	0	1	50.0
REC	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0.0
TMP	23	0	0	4	0	11	7	0	1	1	0	0	877	5	51	89.1
CORE	4	0	2	1	0	0	5	0	0	6	0	0	8	8303	372	95.6
NONE	31	1	9	25	0	45	43	3	1	5	0	0	54	348	–	–

(c) Modifier arguments confusion matrix.

Figure 3.14: Performance measures for joint model using all features (AllJoint). Results are on Section 23 using gold standard parse trees and argument-based scoring.

Rank	1	2	3	4	5	6	7	8	9	10
Chosen	84.1	8.6	2.7	1.6	0.8	0.4	0.5	0.3	0.2	0.4

Figure 3.15: Percentage of test set propositions for which each of the top ten assignments from the LOCAL model was selected as best by the joint model ALLJOINT.

significantly, and also there is a large decrease in the confusions of NONE with ARG1. There is generally a slight increase in F-Measure on modifier labels as well, but the performance on some of the modifier labels has gone down. This makes sense because our joint features are targeted at capturing the dependencies among core arguments. There may be useful regularities for modifier arguments as well, but capturing them may require different joint feature templates.

Figure 3.15 lists the frequency with each each of the top k assignments from the LOCAL model was ranked first by the re-ranking model ALLJOINT. For example, for 84.1% of the propositions, the re-ranking model chose the same assignment that the LOCAL model would have chosen. The second best assignment according to the LOCAL model was promoted to first 8.6% of the time. The Figure shows statistics for the top ten assignments only. The rest of the assignments, ranked 11 through 15, were chosen as best by the re-ranking model for a total of 0.3% of the propositions.

The labeling of the tree in Figure 3.2 is a specific example of the kind of errors fixed by the joint models. The local classifier labeled the first argument in the tree as ARG0 instead of ARG1, probably because an ARG0 label is more likely for the subject position.

3.7 Automatic Parses

We now evaluate our models when trained and tested using automatic parses produced by Charniak's parser. The PropBank training set Sections 2–21 is also the training set of the parser. The performance of the parser is therefore better on the training set. When the constituents of an argument do not have corresponding constituents in an automatically produced parse tree, it will be very hard for a model to get the semantic role labeling correct. For constituent-based scoring, these argument

Set	Constituents	Propositions
Training	2.9%	7.4%
Development	7.1%	17.3%
Test	6.2%	15.7%

Figure 3.16: Percentage of argument constituents that are not present in the automatic parses of Charniak's parser. **Constituents** shows the percentage of missing constituents and **Propositions** shows the percentage of propositions that have missing constituents.

constituents cannot possibly be guessed correctly by the labeling system and there is an upper bound on performance determined by the parser error. For argument-based scoring the system can theoretically guess the correct set of words by labeling any existing constituents that dominate a subset of the argument words. However, we found that argument-based measures are usually lower than constituent-based ones and in practice the same or lower upper bound applies to this scoring setting.

Figure 3.16 shows the percentage of argument constituents that are missing in the automatic parse trees, produced by Charniak's parser. The percentage of propositions that have at least one missing argument constituent indicates an upper bound on the whole frame accuracy achievable by labeling Charniak parse trees (for constituent-based scoring). We can see that the percentage of missing constituents is quite high and that the upper bound for whole frame accuracy is far from 100% – it is 84.3% for the test set.

We report local and joint model results using argument-based scoring in Figures 3.17(a) and 3.17(b) respectively. We also compare the confusion matrices of the local and joint models, ignoring the confusions among modifier argument labels (CoarseARGM setting) in Figure 3.18. The error reduction of the joint over the local model is 10.3% in CORE arguments F-Measure and 8.3% in ALL arguments F-Measure.

3.7.1 Using Multiple Automatic Parse Guesses

Semantic role labeling is very sensitive to the correctness of the given parse tree as the above results show. If an argument does not correspond to a constituent in a

Task	CORE		CoarseARGM		ALL	
	F1	Acc.	F1	Acc.	F1	Acc.
Id	82.2	67.3	81.7	60.2	81.7	60.2
Cls	98.0	97.1	98.0	96.1	95.7	91.7
Id&Cls	80.6	65.6	80.1	58.1	78.2	55.3

(a) Summary performance of local model.

Task	CORE		CoarseARGM		ALL	
	F1	Acc.	F1	Acc.	F1	Acc.
Id	84.2	70.5	83.4	64.0	83.4	64.0
Cls	98.0	97.3	98.1	96.4	95.9	92.0
Id&Cls	82.6	69.1	81.8	62.0	80.0	59.1

(b) Summary performance of joint model.

Figure 3.17: Comparison of local and joint model results on Section 23 using Charniak's automatic parser and argument-based scoring.

parse tree or a constituent exists but is not attached or labeled correctly, our model will have a very hard time guessing the correct labeling.

One way to address this problem is to consider alternative parses. Recent releases of the Charniak parser (Charniak, 2000) have included an option to provide the top k parses of a given sentence according to the probability model of the parser. We use these alternative parses as follows: Suppose $t_1, \ldots, t_k$ are trees for sentence s with probabilities $P(t_i|s)$ given by the parser. Then for a fixed predicate v, let L_i denote the best joint labeling of tree t_i, with score $score_{SRL}(L_i|t_i)$ according to our final joint model. Then we choose the labeling L which maximizes:

$$\text{argmax}_{i \in \{1,\ldots,k\}} \beta \log P(t_i|S) + score_{SRL}(L_i|t_i)$$

Figure 3.19 shows summary results for the test set when using top ten parses and the joint model. The weighting parameter for the parser probabilities was $\beta = 1$. We didn't experiment extensively with different values of β. Preliminary experiments showed that considering 15 parses was a bit better, and considering top 20 was a bit

	Guessed								
Correct	ARG0	ARG1	ARG2	ARG3	ARG4	ARG5	ARGM	NONE	F-Measure
ARG0	2710	23	3	0	0	0	5	446	86.1
ARG1	62	3423	14	0	0	0	10	855	79.7
ARG2	6	23	610	0	0	0	7	291	69.0
ARG3	1	4	2	62	1	0	4	60	55.4
ARG4	0	1	0	0	50	0	1	22	73.5
ARG5	0	0	0	0	0	2	0	3	57.1
ARGM	2	9	8	3	0	0	2446	778	78.7
NONE	330	740	194	25	11	0	497	–	–

(a) CoarseARGM confusion matrix of local model.

	Guessed								
Correct	ARG0	ARG1	ARG2	ARG3	ARG4	ARG5	ARGM	NONE	F-Measure
ARG0	2808	32	4	0	0	0	4	339	88.2
ARG1	34	3579	18	0	0	0	9	724	81.9
ARG2	5	34	653	1	1	0	5	238	70.6
ARG3	1	5	4	72	1	0	2	49	59.3
ARG4	0	0	1	0	54	0	0	19	73.0
ARG5	0	0	0	0	0	4	0	1	88.9
ARGM	2	15	10	2	1	0	2519	697	79.5
NONE	329	712	223	34	17	0	552	–	–

(b) CoarseARGM confusion matrix of joint model.

Figure 3.18: CoarseARGM argument confusion matrices for local and joint model using Charniak's automatic parses.

Task	CORE		CoarseARGM		ALL	
	F1	Acc.	F1	Acc.	F1	Acc.
Id	85.2	72.1	84.0	65.7	84.0	65.7
Cls	98.2	97.4	98.1	96.5	96.2	92.6
Id&Cls	83.6	70.8	82.5	63.9	80.8	61.2

Figure 3.19: Performance of the joint model using top ten parses from Charniak's parser. Results are on Section 23 using argument-based scoring.

worse.

3.7.2 Evaluation on the CoNLL 2005 Shared Task

The CoNLL 2005 evaluation ensures results obtained by different teams are evaluated in exactly the same way. To provide a basis for fair comparison with other work, we report results on this data as well.

The CoNLL 2005 data is derived from Propbank version I, which is the latest official release from 2005. The results we have been reporting in the previous sections used the February 2004 data. In Propbank I, there have been several changes in the annotation conventions, as well as error fixes and addition of new propositions. There was also a change in the way PP arguments are annotated – in the February 2004 data, some PP arguments are annotated at the head NP child, but in Propbank I, all PP arguments are annotated at the PP nodes. In order to achieve maximal performance with respect to these annotations, it would probably be profitable to change the feature definitions to account for the changes. However, we did no adaptation of the features.

The training set consists of the annotations in Sections 2 to 21, the development set is section 24 (Devset), and one of the test sets is section 23 (Test WSJ). The other test set is from the Brown corpus (Test Brown). The CoNLL annotations distinguish referring arguments, of the form R-ARGX, as discussed briefly in §3.4. The evaluation measure compares labeled sets of words, like our argument-based measures, with the difference that R-ARGX is a separate argument label. For example, for the sentence: "The book that I bought yesterday was good.", the correct labeling would be: "[The

book]$_{\text{ARG1}}$ [that]$_{\text{R-ARG1}}$ [I]$_{\text{ARG0}}$ *bought* [yesterday]$_{\text{ARGM-TMP}}$ was good".

Our approach to dealing with referring arguments and for deciding when multiple constituents labeled the same are part of the same argument was to label constituents with only the set of argument labels and NONE and then map some of these labels into referring or continuation labels. The rule for converting a label of ARGX into R-ARGX was the following: the label is referring if and only if the phrase type of the constituent starts with "WH". The rule for deciding when to add continuation labels was the same as for our systems for the February 2004 data – a constituent label becomes continuing if and only if it is a core argument label and there is another constituent with the same core argument label to the left. Therefore, for the CoNLL 2005 shared task we employ the same semantic role labeling system, just using a different post-processing rule to map to CoNLL-style labelings of sets of words.

We tested the upper bound in performance due to our conversion scheme in the following way – take the gold standard CoNLL annotations for the development set (including referring and continuing labels), convert those to basic argument labels of the form ARGX, then convert the resulting labeling to CoNLL-style labeling using our rules to add the referring and continuing annotations back in. The F-Measure obtained was 98.99.

Figure 3.20 shows the performance of the local and joint model on one of the CoNLL test sets – Test WSJ (section 23)– when using gold standard parse trees. Performance on gold standard parse trees was not measured in the CoNLL 2005 shared task, but we report it here to provide a basis for comparison with the results of other researchers.

Next we present results using Charniak's automatic parses on the development and two test sets. We present results for the local and joint models using the max-scoring Charniak parse tree. Additionally, we report results for the joint model using top five Charniak parse trees according to the algorithm described in §3.7.1.

The performance measures reported here are higher than the results of our submission in the CoNLL 2005 shared task (Haghighi et al., 2005), because of two changes.

Model	Test WSJ	
	F1	Acc.
Local	87.61	71.41
Joint	89.89	75.64

Figure 3.20: Results on the CoNLL WSJ Test set, when using gold standard parse trees.

Model	Devset		Test WSJ		Test Brown	
	F1	Acc.	F1	Acc.	F1	Acc.
Local	75.13	51.60	77.03	53.24	65.37	34.83
Joint	77.20	55.79	78.68	56.43	67.67	38.68

Figure 3.21: Results on the CoNLL dataset, when using Charniak automatic parse trees as provided in the CoNLL 2005 shared task data.

One was changing the rule that maps to continuing arguments to only add continuation labels to core argument labels; in the previous version the rule added continuation labels to all repeated labels. Another was finding a bug in the way the sentences were passed in as input to Charniak's parser.

Interestingly, the Charniak parses provided as part of the CoNLL shared task data, uniformly ignore the distinction between forward and backward quotes and all quotes are backward. For example, both quotes in " phrase " are tagged and regarded as backward or closing quotes by the parser. This may be due to an incorrect input to the parser. This may seem to be a minor detail but accounts for one point in F-Measure on semantic role labeling and would have made the difference for winning the competition for one of the top systems.

We first present results of our local and joint model using the parses provided as part of the CoNLL 2005 data (and having wrong forward quotes) in Figure 3.21. We then report results from the same local and joint model, and the joint model using top five Charniak parses, where the parses have correct representation of the forward quotes in Figure 3.22. If the Charniak parser was passed in as input a string of the form " phrase ", it split the forward quotes into two separate tokens. We therefore preprocessed the input string by concatenating all forward quotes to the following word, like this: "phrase ", which led to proper behavior on the part of the parser.

Model	Devset		Test WSJ		Test Brown	
	F1	Acc.	F1	Acc.	F1	Acc.
Local	75.80	53.05	78.00	55.31	65.55	35.70
Joint	77.93	57.20	79.71	58.65	67.79	39.43
Joint top 5	78.64	58.65	80.32	60.13	68.81	40.80

Figure 3.22: Results on the CoNLL dataset, when using Charniak automatic parse trees, version of the Charniak parser from May 2005 with correct treatment of forward quotes.

For these results we used the version of the Charniak parser from May 4, 2005. The results were very similar to the results we obtained with the version from March 18, 2005. Note that the parser from May 4 we used does not incorporate the new re-ranking model of Charniak and Johnson (Charniak and Johnson, 2005), which improves upon (Charniak, 2000) significantly. The new re-ranking parser is also freely available, but we did not use it for these experiments.

For comparison, the system we submitted to CoNLL 2005 had an F-Measure of 78.45 on the WSJ Test set. The winning system (Punyakanok et al., 2005) had an F-Measure of 79.44 and our current system has an F-Measure of 80.32. For the Brown Test set, our submitted version had an F-Measure of 67.71, the winning system had 67.75, and our current system has 68.81.

Figure 3.23 shows the per-label performance of our joint model using top five Charniak parse trees, on the Test WSJ test set.

3.8 Conclusions

Reflecting linguistic intuition and in line with current work, we have shown that there are substantial gains to be had by jointly modeling the argument frames of verbs. This is especially true when we model the dependencies with discriminative models capable of incorporating long-distance features.

For further improving performance in the presence of perfect syntactic parses, we see the biggest avenues for improvement as the following:

	Precision	Recall	$F_{\beta=1}$
Overall	81.90%	78.81%	80.32
A0	88.37%	88.91%	88.64
A1	81.50%	81.27%	81.38
A2	73.44%	68.74%	71.01
A3	75.00%	55.49%	63.79
A4	74.74%	69.61%	72.08
A5	100.00%	80.00%	88.89
AM-ADV	64.86%	54.35%	59.14
AM-CAU	67.92%	49.32%	57.14
AM-DIR	55.74%	40.00%	46.58
AM-DIS	81.69%	75.31%	78.37
AM-EXT	65.00%	40.62%	50.00
AM-LOC	66.45%	56.75%	61.22
AM-MNR	62.50%	56.69%	59.45
AM-MOD	98.00%	98.00%	98.00
AM-NEG	97.83%	97.83%	97.83
AM-PNC	54.22%	39.13%	45.45
AM-PRD	100.00%	20.00%	33.33
AM-REC	0.00%	0.00%	0.00
AM-TMP	80.12%	72.31%	76.02
R-A0	93.15%	91.07%	92.10
R-A1	80.50%	82.05%	81.27
R-A2	61.54%	50.00%	55.17
R-A3	0.00%	0.00%	0.00
R-A4	0.00%	0.00%	0.00
R-AM-ADV	0.00%	0.00%	0.00
R-AM-CAU	100.00%	50.00%	66.67
R-AM-EXT	0.00%	0.00%	0.00
R-AM-LOC	76.92%	47.62%	58.82
R-AM-MNR	50.00%	33.33%	40.00
R-AM-TMP	74.00%	71.15%	72.55
V	97.32%	97.32%	97.32

Figure 3.23: Per-label performance of joint model using top five Charniak automatic parse trees on the Test WSJ test set.

- Improving the identification of argument nodes, by better handling of long-distance dependencies, i.e., incorporating models which recover the trace and null element information in Penn Treebank parse trees, such as (Levy and Manning, 2004).

- Improving the accuracy on modifier labels, by improving the knowledge about the semantic characteristics of specific words and phrases, i.e., improving lexical statistics.

- Better handling of multi-constituent arguments; our current model uses a simple rule in a post-processing step to decide which constituents labeled the same are part of the same argument, but this could be done more intelligently by the machine learning model.

- Building a joint model which does not use re-ranking over a local model, but a more exact dynamic programming or approximation technique to deal with computational complexity.

Since perfect syntactic parsers do not yet exist and the major bottleneck to the performance of current semantic role labeling systems is syntactic parser performance, the more important question is how to improve performance in the presence of parser errors. We explored a simple approach of choosing from among the top k parses from Charniak's parser which resulted in an improvement. There should be better ways of integrating syntactic and semantic parsing and work on alternative approaches already exists (Punyakanok et al., 2005; Yi and Palmer, 2005). This is a very promising line of research.

Chapter 4

Random Walks for Estimating Word Dependency Distributions

In the previous chapter we showed that it is very useful to increase the lexicalization of structural features in syntactic parse selection. More broadly, since every word has its own different semantics and often very specific syntax, the more reliable statistics we can gather that are specific to individual words, the better our models will perform.

This chapter addresses the problem of estimating word dependency distributions – distributions over words, conditioned on words. For example $P_{OBJ}(n|v)$ is a dependency distribution over object nouns for verbs. Parts of this work were reported in (Toutanova et al., 2004a).

4.1 Introduction

Word dependency or co-occurrence probabilities are needed in many natural language tasks. This includes lexicalized parsing, building language models, word sense disambiguation, and information retrieval. A touchstone problem in parsing and one where knowledge about word dependencies is crucial is the problem of deciding the attachment of preposition phrases (PPs). For example, in the sentence: *He broke the window with a hammer*, the prepositional phrase *with a hammer* could either modify

the verb *broke*, and thus mean that the *hammer* was the instrument of the breaking event, or it could modify the noun *window* and thus mean that the *window* perhaps had a stained glass rendition of a *hammer* in it. People immediately recognize the more plausible meaning using their world knowledge, but this knowledge is not readily available to parsers. Previous research has shown that by using statistics of lexical co-occurrences, much higher accuracy can be achieved in comparison to approaches that only look at structure (such as preferring attachment to a verb or the closer word, etc.) (Hindle and Rooth, 1993).

However, it is difficult to estimate word dependency probabilities because of the extreme sparseness of data for individual words, and even more so for word pairs, triples, and so on. For instance, (Bikel, 2004) shows that the parser of (Collins, 1999) is able to use bi-lexical word dependency probabilities[1] to guide parsing decisions only 1.5% of the time (28.8% for its most likely guess); the rest of the time, it backs off to condition one word on just phrasal and part-of-speech categories. Even though the 28.8% number is more relevant, it is still very small. If a system could be built with reasonably accurate knowledge about dependency probabilities between all words, one would expect the performance gains on many tasks to be substantial.

Sophisticated back-off and interpolation methods have been developed for language modeling (Goodman, 2001). (Dagan et al., 1999) showed that performance on zero-count events can be greatly improved if the model includes estimates based on distributional similarity. Other kinds of similarity among words have also been used to reduce sparseness. For instance, stemming words is a very traditional way of somewhat lessening sparseness, and resources like WordNet (Miller, 1990) have been used in many natural language models.

All of these ways of using associations and similarities between words to predict the likelihood of unseen events have their advantages. Symbolic knowledge bases, such as WordNet, have the advantage of being based on abundant world knowledge and human intuition, but have the disadvantages of having incomplete coverage and being non-probabilistic. Using stemming or lemmatized words has been helpful for

[1]Bi-lexical probabilities include two words, one in the conditioning context and one in the future, in addition to possibly other variables, for example, $P(salad|eat, V, VP)$.

reducing sparseness in some problems, and slightly harmful in others (Hull, 1996).

We propose a method for combining these information sources that induces a distribution over words by learning a Markov Chain (random walk) model, where the states correspond to words, such that its stationary distribution is a good model for a specific word-distribution modeling task. The idea of constructing Markov Chains whose stationary distributions are informative has been seen in several other applications, such as the Google PageRank algorithm (Brin and Page, 1998), some HITS (Kleinberg, 1998)-like link analysis algorithms (Ng et al., 2001), and for query expansion in IR (Lafferty and Zhai, 2001). Our work is distinguished from these approaches in that rather than using a carefully hand-picked Markov Chain, we will automatically *learn* the parameters for the random walk. This allows us to construct Markov Chains with many more parameters, that are much richer in structure and of significantly greater complexity than seen in other applications. In doing so, we can also allow our model to learn to exploit diverse knowledge sources such as WordNet, morphology, and various features of words derived from dependency relations; all of these simply become additional "features" made available to the random walk learning algorithm. The proposed techniques are general and can be applied to other problem domains, such as the web, citation, and clickstream data.

After completing this work, we became aware that an algorithm that estimates a distribution of interest as the stationary distribution of a Markov Chain, whose parameters are learned from data, was previously developed by Jason Eisner in (Eisner, 2001; Eisner, 2002). The application was estimating the weights of sub-categorization frames of verbs and is thus quite different from our application of incorporating multiple knowledge sources for inducing word dependency distributions. We discuss the relation to that work in §4.7.

In this work, we choose PP attachment as a classic problem where lexical dependency distribution are important, and show how random walk methods can be applied to this problem. The focus of our work is on estimation of lexical distribution and PP attachment is an illustration of doing that well.

4.2 Markov Chain Preliminaries

We briefly review the basic properties of Markov Chains (MC). For a more detailed treatment, see, e.g., (Taylor and Karlin, 1998; Brémaud, 1999). In the following, we will use P to denote a probability value and p to denote a probability mass function.

A *stochastic process* $\{S_t\}$ is a family of random variables, where t ranges over an index set T. A *Markov process* $\{S_t\}$ is a stochastic process over a totally ordered index set satisfying *the Markov property*: for any indices $k < t < m$, S_m is independent of S_k given S_t. A *discrete-time Markov Chain* is a Markov process whose state space $\mathcal{S}$ is finite or countable and whose index set T is the set of natural numbers $T = (0, 1, \ldots)$. The Markov property for a discrete-time Markov Chain can be written as:

$$P(S_t = j | S_0 = i_o, \ldots, S_{t-2} = i_{t-2}, S_{t-1} = i) = P(S_t = j | S_{t-1} = i)$$

$$\forall t, j, i, i_0, \ldots, i_{t-2}$$

The Markov Chain is *time-homogeneous* or *stationary*, if the transition probabilities do not depend on the time t. More formally:

$$\forall t, i, j : P(S_t = j | S_{t-1} = i) = P(S_1 = j | S_0 = i)$$

A discrete time stationary Markov Chain over a set of states $\mathcal{S}$ is specified by an **initial distribution** $p_0(S)$ over $\mathcal{S}$, and a set of **state transition probabilities** $p(S_t|S_{t-1})$.The state transition probabilities can be represented by a matrix $\mathbf{P}$, whose entries are $P^{ij} = P(S_t = j|S_{t-1} = i)$. A Markov Chain defines a distribution over sequences of states, via a generative process in which the initial state S_0 is first sampled from according to p_0, and then states S_t (for $t = 1, 2, \ldots$) are sampled in order according to the transition probabilities. From now on, when we mention a Markov Chain (MC), we will mean a discrete-time stationary Markov Chain.

$P(S_t = s)$ denotes the probability that the random variable S_t has value s. This probability can also be referred to as the probability that the MC is in state s at time t. We can compute the probability distribution $p(S_t)$ at time t using the initial

distribution and the state transition probabilities in the following way:

$$\begin{aligned} p(S_0) &= p_0 \\ p(S_1) &= p_0 P \\ p(S_2) &= p_0 P P \\ \vdots &= \vdots \\ p(S_t) &= p_0 P^t \end{aligned}$$

A MC has a *limiting distribution* π if, for every state s, the chain started at s converges to the same distribution π. Formally,

$$\forall s \lim_{t \to \infty} p(S_t | S_0 = s) = \pi$$

A MC has a *stationary distribution* π, if the chain stays in π if it is started according to π. More formally π is a stationary distribution if and only if:

$$\pi = \pi P$$

If a MC has a limiting distribution π, it can be shown that it has a unique stationary distribution which is also π (Taylor and Karlin, 1998).

The MCs used in (Brin and Page, 1998; Ng et al., 2001) have the property that on each step, there is a probability $\gamma > 0$ of resetting according to the initial state distribution p_0. Thus, the state transition probabilities can be written

$$p(S_t | S_{t-1}) = \gamma p_0(S_t) + (1 - \gamma) p'(S_t | S_{t-1}) \tag{4.1}$$

for some appropriate p'. This ensures that the MC has a limiting distribution, and therefore it also has a unique stationary distribution (Brémaud, 1999). Additionally, in practice this property also prevents the chain from getting stuck in small loops (Brin and Page, 1998).

Given a MC $S_0, S_1, \ldots$, given by the initial distribution p_0 and the state transition probabilities as specified in Equation 4.1, we can construct another MC $S'_0, S'_1, \ldots$ with the initial state S'_0 distributed according to p_0, and state transitions given by the p' in Equation (4.1). It is straightforward to show that

$$\pi(s) = \gamma \sum_{t=0}^{\infty} (1-\gamma)^t P(S'_t = s) \tag{4.2}$$

where π here is the limiting distribution of the *original* MC $S_0, S_1, \ldots$. Equation 4.2 can be used to efficiently compute π. Also, because the quantity $(1-\gamma)^t$ rapidly becomes very small, when computing π, this sequence may be truncated after the first few (on the order $1/\gamma$) terms without incurring significant error.

Equation (4.2) gives a useful alternative view of π. Consider a random process in which the state S_0 is initialized according to p_0. On each time step t, with probability γ we "stop" the chain and output the current state S_t; and with probability $1-\gamma$, we till take a state transition step and sample S_{t+1} according to the transition probabilities $p'(S_{t+1}|S_t)$. This process is continued until the chain is stopped and a state is output. Because the number of steps T taken in the chain until it is stopped is distributed according to a geometric distribution with parameter $(1-\gamma)$, we can see using Equation (4.2) that the random state output by this process will also be distributed according to π.

For the application considered in this work, it will be useful to consider a generalization of this random process. Specifically, we will construct a MC where, once we have decided to stop the MC (which happens with probability γ on each step), we will allow the state to transition one final time according to a new set of transition probabilities $p''(S_{t+1}|S_t)$ (different from the transition probabilities used in the earlier steps of the walk), and finally output S_{t+1}. Note that if $p''(S_{t+1}|S_t) = 1$ iff $S_{t+1} = S_t$, this reduces to the simpler type of random walk described earlier. In §4.6 we will see how permitting an extra state-transition step at the end allows us to build significantly more expressive models.

4.3 Related Work on Smoothing

.

Dealing with sparsity has been recognized as an extremely important issue in machine learning for natural language processing. A key application in which dealing with sparsity is crucial is language modeling for speech recognition. Many smoothing methods were developed or applied in the context of language modeling. Some examples include Lidstone (Lidstone, 1920), Good-Turing (Good, 1953), Katz (Katz, 1987), Jelinek-Mercer (Jelinek and Mercer, 1980), Kneser-Ney (Kneser and Ney, 1995), and Witten-Bell (Witten and Bell, 1991) smoothing. Such smoothing methods have been applied in other natural language applications as well, such as lexicalized parsing (Collins, 1997; Charniak, 2000).

Other relevant work has used word similarities to derive dependency distributions. Examples include class-based language models (Brown et al., 1992), co-occurrence smoothing (Essen and Steinbiss, 1992), and nearest neighbor-type smoothing based on word similarities (Dagan et al., 1999; Lee, 1999). This work provides the basis of our framework for smoothing based on multiple sources of information. We will show in the discussion section how these smoothing methods are special instances of our framework.

In n-gram language modeling, the probability of a sentence $P(w_1, w_2, \ldots, w_n)$ is decomposed into

$$\prod_{i=1,\ldots,n} P(w_i|w_{i-1}, \ldots, w_{i-n+1})$$

by making an order n Markov independence assumption. The individual n-gram lexical probabilities for the model are estimated using some smoothing method. Empirical studies show that smoothing methods can vary widely in performance, depending on the specific conditions of the task (Chen and Goodman, 1998). The individual smoothing methods differ in the ways they adjust the mass of seen events to reserve mass for unseen ones, and how they use the information from lower-order distributions (containing less conditioning context) to estimate higher-order ones. For a thorough overview of smoothing models for language modeling see, for example, (Chen and

Goodman, 1998; Goodman, 2001).

We briefly review Jelinek-Mercer smoothing here, because it is the basis for our model. Chen and Goodman (1996) found that this method was one of the best-performing under a variety of conditions, and the updated (Chen and Goodman, 1998) also found that it was relatively good, outperformed consistently only by variations of Kneser-Ney (Kneser and Ney, 1995). The form of Jelinek-Mercer smoothing is as follows:

$$\begin{aligned} P_{JM}(w_i|w_{i-1},\ldots,w_{i-n+1}) &= \lambda(w_{i-1},\ldots,w_{i-n+1}) \times \hat{P}(w_i|w_{i-1},\ldots,w_{i-n+1}) + \\ &+ (1-\lambda(w_{i-1},\ldots,w_{i-n+1})) \times P_{JM}(w_i|w_{i-1},\ldots,w_{i-n+2}) \end{aligned} \tag{4.3}$$

After all variables are removed from the context, the recursion is terminated with the uniform distribution $\frac{1}{V}$, where V is the vocabulary size. Here $\hat{P}$ denotes a relative frequency probability value, estimated from a training set. We see that the estimated distribution conditioned on $n-1$ previous words is obtained using the estimated distribution conditioned on the previous $n-2$ words, interpolated with a relative frequency estimate of the full distribution. The interpolation parameters λ depend on the particular $n-1$ word context, to allow us to give more weight to the relative frequency estimate in some cases. For example, if we have seen very large amounts of data for a particular context, we can be more confident in the relative frequency (maximum likelihood) estimate.

In practice, we cannot learn interpolation weights specific to every context. Good performance is obtained if the interpolation parameters for contexts with similar number of occurrence in the training data are binned into equivalence classes and the parameters for those classes are tied. The interpolation parameters are fit to maximize the likelihood of held-out data (Jelinek and Mercer, 1980). In §4.5.1, we will see that our baseline PP attachment model uses a very similar form of smoothing.

4.3.1 Estimating Distributions as Limiting Distributions of Markov Chains

A well-known example of using the stationary distribution of a Markov Chain is the Page-Rank algorithm (Brin and Page, 1998). The page rank of a web-page w is the estimated frequency with which a random surfer visits the web page. It is defined as the probability of w according to the stationary distribution of the following Markov Chain, defined using the web graph. The initial distribution of the Markov Chain p_0 is uniform over all web pages. The state transition probabilities from a page w are defined as interpolation of the uniform distribution over web pages p_0 and a uniform distribution over all web pages that the page w links to. Thus the form of the transition matrix is as described in Equation 4.1. The parameter in the interpolation can vary, but everything else is fixed.

Another example, in the domain of estimating word distributions, is a language model for query expansion in IR (Lafferty et al., 2001). The goal is to estimate a translation probability $P(q|w)$ of a query word q given another word w for query expansion. This is estimated as the limiting distribution of a Markov Chain started at w and transitioning according to a mixture of the initial distribution and a distribution obtained directly by using the word-document frequencies. Thus the Markov Chain is of the same form as in Equation 4.1. There are no trainable parameters of the model (the mixture weight was fixed to 0.5). This method is very similar to co-occurrence smoothing (Essen and Steinbiss, 1992).

The work most closely related to the present work is (Eisner, 2001; Eisner, 2002), which uses the limiting distribution of a Markov Chain, whose transition parameters are learned automatically from data. The Markov Chain is used to estimate a conditional distribution over subcategorization frames given verbs $p(subcat|verb)$. The transition matrix is parameterized using a log-linear model over a set of features indicating relatedness among states of the Markov Chain (subcategorization frames). We discuss the similarities and differences to this approach in §4.7.

4.4 The Prepositional Phrase Attachment Task

Prepositional phrases are one of the major sources of ambiguity in English. Following most of the literature on prepositional phrase (PP) attachment (e.g., (Hindle and Rooth, 1993; Brill and Resnik, 1994; Collins and Brooks, 1995; Stetina and Nagao, 1997; Harabagiu and Pasca, 1999; Pantel and Lin, 2000)), we focus on the most common configuration that leads to ambiguities: V NP PP. Here, we are given a verb phrase with a following noun phrase and a prepositional phrase. The goal is to determine if the PP should be attached to the verb or to the object noun phrase. For example, in the sentence:

(4.4) *Never* $[hang]_V$ $[a\ painting]_{NP}$ $[with\ a\ peg]_{PP}$.

the prepositional phrase *with a peg* can either modify the verb *hang* or the object noun phrase *the painting*. Here, clearly, *with a peg* modifies the verb *hang*.

Previous work has shown the central (but not exclusive) role played by the head words of phrases in resolving such ambiguities, and we follow common practice in representing the problem using only the head words of these constituents and of the NP inside the PP. Thus the example sentence is represented as the following quadruple:

v:hang n_1:painting p:with n_2:peg

The task is thus to decide the attachment Att given the four head words – v, n_1, p, n_2. The variable Att has as value either va (for verbal attachment) or na (nominal/noun attachment). Since all features are words, and the two types of attachment correspond to different dependency relations, this disambiguation problem is a good testing ground for evaluation of models for estimating word dependency distributions.

The prepositional phrase ambiguity problem is naturally much harder when multiple prepositional phrases are present in the sentence and when other candidate attachment sites are possible. There is some work on statistical classification models

for more general settings (Merlo et al., 1997). Conversely, the binary PP attachment problem is easier when more context from the sentence is used (Olteanu and Moldovan, 2005). However, the limited context binary case we concentrate on here is a good starting point for addressing such ambiguity problems. Furthermore, if good probability estimates for lexical relations are obtained, they can be easily incorporated into models dealing with the more general ambiguities.

4.4.1 Dataset

We work with the Penn Treebank Wall Street Journal data (Ratnaparkhi et al., 1994), which consists of four-tuples of head words and a specification of the type of attachment of the form $[v, n_1, p, n_2, Att]$. There are 20,801 samples in the training set, 4,039 in the development set, and 3,097 samples in the test set. The dataset was originally supplied by IBM. The examples were extracted from the Wall Street Journal Penn Treebank (Marcus et al., 1993). We refer to this dataset as the IBM dataset. We preprocessed the original data by lower-casing the verbs and prepositions, and by substituting all digits with the X symbol. In this way many word tokens are unified — for example all four-digit numbers (usually denoting years) are represented by XXXX. Similar preprocessing was done in (Collins and Brooks, 1995). Figure 4.4.1 shows summary statistics of the training, development, and test sets. The number of examples is shown, as well as the number of word types for each of the four headwords. For the development and test sets, the number of tokens and types unseen in training is also shown. The statistics are presented for the preprocessed data.

This same data has been used by many other researchers (Ratnaparkhi et al., 1994; Collins and Brooks, 1995; Stetina and Nagao, 1997). The preprocessing may possibly differ across systems but the difference in performance due to preprocessing is very small as we will show in §4.5.1. A previous study of human performance suggests an upper bound on attachment accuracy, given just the four-tuple of head words, of 88.2% (Ratnaparkhi et al., 1994). Therefore it is plausible to accept a Bayes error of about 10% for this task.

The upper bound rises to 93.2% when the whole sentence is presented to human

	Training				Devset				Test			
	types		tokens		types		tokens		types		tokens	
	all	unkn	all	unkn	all	unkn	all	unkn	all	unkn	all	unkn
V	3,243	–	20,801	–	1281	253	4,039	279	1123	231	3,097	257
N_1	4,172	–	20,801	–	1547	438	4,039	503	1261	276	3,097	303
N_2	5,146	–	20,801	–	1762	518	4,039	563	1294	318	3,097	353
P	66	–	20,801	–	50	3	4,039	4	52	2	3,097	3

Figure 4.1: Dataset characteristics.

annotators. A reasonable lower bound is obtained by choosing the most frequent attachment for a given preposition according to the training data. The accuracy of this approach is 72.2%.

4.4.2 Previous Work on PP Attachment

We now review previous work on machine learning approaches for deciding the attachment of the prepositional phrase in the V NP PP configuration, using at most the four head words involved in the ambiguity.

Hindle & Rooth 93

The first statistical approach that explored co-occurrence statistics in data was the unsupervised model of Hindle & Rooth proposed in the early 1990s (Hindle and Rooth, 1993). It was trained and tested on a different dataset than the one we consider in this work. Only three head words were considered V, N_1, and P. Michael Collins (Collins, 1999) re-implemented this model as a supervised one trained on the IBM training set. Its test set performance was 81.3%. The model is similar in a way to our baseline generative model described in §4.5.1. It is a generative model for the triple of head-words and the attachment site: $p(Att, V, N_1, P)$. However, the independence assumptions made are much stricter than the ones we make. The assumed form of the model is as follows:

$$\begin{aligned} p(va, V, N_1, P) &= p(V)p(N_1)p(va|N_1)p(P|V, va) \\ p(na, V, N_1, P) &= p(V)p(N_1)p(na|N_1)p(P|N, na) \end{aligned} \tag{4.5}$$

Since the probabilities $p(V)$ and $p(N_1)$ participate in the joint probability of the tuple

under both noun and verb attachment, they can be safely ignored for classification purposes. We can see then that the model is based on estimation of probabilities involving a single content word ($p(Att|N_1)$) or a single word and the preposition ($p(P|V, va)$, $p(P|N_1, na)$). We will see later that it is possible to do much better by using estimates for pairs, triples, and quadruples of words.

Collins & Brooks 95

A better performing model using more lexical information was proposed by Collins & Brooks in 1995 (Collins and Brooks, 1995). This is a supervised classification model. It computes statistics based on the four head-words (V, N_1, P, N_2) and the attachment site Att. The model is inspired by Katz back-off estimation (Katz, 1987). It estimates the probability distribution $\tilde{p}(Att|V, N_1, P, N_2)$ by looking at the counts of occurrence of the the four head words with the two attachment types. The estimation procedure is similar to a nearest neighbor algorithm in that it uses the counts of the most specific context found. For example, if the test quadruple (v, n_1, p, n_2) has been seen in training, the algorithm uses a relative frequency estimate based on that context, i.e. $\hat{P}(Att|v, n_1, p, n_2)$. If not, the algorithm tests if any of the triples involving the preposition – (v, n_1, p), (v, n_2, p), or (n_1, n_2, p) – has been seen. If so, the estimate is based on their pooled counts:

$$\tilde{P}(Att = na|v, n_1, p, n_2) = \frac{c(na, v, n_1, p) + c(na, v, n_2, p) + c(na, n_1, n_2, p)}{c(v, n_1, p) + c(v, n_2, p) + c(n_1, n_2, p)}$$

where $c(x)$ denotes the count of the event x in the training data. If neither of these word triples has been seen, the algorithm backs off to form an estimate based on pairs, and if no pairs have been seen, it backs off further.

The overall performance of this model using the same standard training set that we use (described in §4.4.1) was 84.1%. The model is simple to understand and implement and has comparable performance to the best previously described machine-learned models derived from the training data of word quadruples. We use this model as the major comparison point from previous work in Sections 4.5.1 and 4.6.4.

A major observation about the performance of the algorithm reported in (Collins,

1999) was that the performance of the algorithm was quite good for examples where a triple or the full quadruple was seen – 90.15% accuracy. However, only 26.5% of the test cases fell into that category. When only a pair of words was seen in the training data, the performance dropped to 82.7%. Therefore, it is crucial to improve the estimates for unseen events, and this is the goal of our work.

Other machine learning methods

Early work on machine learned models for PP attachment includes, for example, (Ratnaparkhi et al., 1994) (a maximum entropy model) and (Brill and Resnik, 1994) (a transformation-based learning model). The accuracies obtained on the test set were below 82%. Subsequent work used other machine learning algorithms such as nearest neighbors (Jakub Zavrel and Veenstra, 1997) and boosting (Steven Abney, 1999). Both algorithms reported an accuracy of 84.4%. The nearest neighbor approach (Jakub Zavrel and Veenstra, 1997) is similar in spirit to our work because it defined a similarity measure between word quadruples based on a similarity measure between words. The similarity measure between words was derived using co-occurrence statistics in a large corpus of text. Thus this measure incorporates a knowledge source which is separate from the training data. We will see how our random walk models can incorporate multiple such sources of knowledge.

The best performing data-driven algorithm, using the four input variables and the same training and test sets with no other sources of information, is a Support Vector Machine model by (Vanschoenwinkel and Manderick, 2003). Its accuracy on the test set is 84.8%. The input features for the SVM are conjunctions of the four head-words. The different conjunctions have weights derived by an information gain criterion.

Stetina & Nagao 97 (Stetina and Nagao, 1997)

The best performing algorithm for PP attachment learned from the same training set and tested on the same test set is a model by Stetina & Nagao (Stetina and Nagao, 1997), with accuracy of 88.1%, which is indistinguishable from the 88.2% human performance. The algorithm uses unsupervised word sense disambiguation into WordNet senses (Miller, 1990) for the training and test sets of PP attachment quadruples. It induces decision trees for the classification decision, using as features

the senses of words and their hypernyms. It may seem that after this result no more work is needed on this version of the PP attachment task. Nevertheless, this algorithm has the drawback that it does not provide word dependency distributions and is thus harder to generalize to more general PP attachment ambiguities or other kinds of syntactic ambiguities. In lexicalized syntactic parsing, we need to have probability estimates for every head-dependent relationship, such as verb-object, verb-subject, verb-adverb, noun-adjective, and noun-noun, in order to assign a score to an entire parse tree.

4.5 The PP Attachment Model

We start by building a generative model for the probability distribution of the sequence of four head words and the attachment site $p(V, N_1, P, N_2, Att)$, where V is a verb, P a preposition, and N_1 and N_2 are the two head nouns involved in the attachment problem. The joint model is built up from word-dependency distributions of the same or more general kind as those common in lexicalized parsing models (Collins, 1997; Charniak, 2000). Using a model for this joint distribution, we can compute the conditional distribution $p(Att|V, N_1, P, N_2)$ and use that to predict the more likely attachment type.

The generative model is illustrated as a Bayesian network in Figure 4.2. No independence assumptions are illustrated in the figure, because we make only context-specific independence assumptions. These are that given a verbal attachment, the second noun is independent of the first noun, and that given a nominal attachment, the second noun is independent of the verb. They determine the following special form for the conditional probability table for the variable N_2 (as earlier, we use lower case p to denote a probability distribution and capital P to denote a probability value):

$$\begin{aligned} p(N_2|va, P, V, N_1) &= p(N_2|va, P, V) \\ p(N_2|na, P, V, N_1) &= p(N_2|na, P, N_1) \end{aligned} \tag{4.6}$$

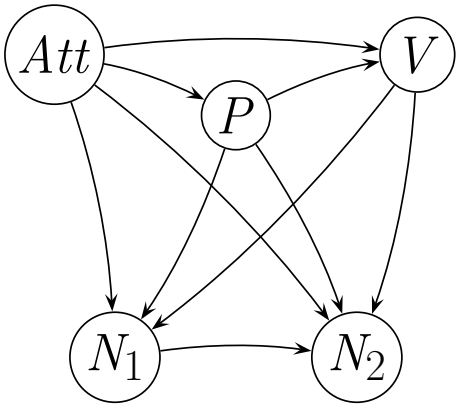


Figure 4.2: Bayesian network of the PP attachment generative model over the four head word variables and the attachment class variable. Only context-specific independence assumptions are made for the variable N_2 described in Equation 4.6.

This context specific independence assumption is natural, because once the attachment site is known, it makes sense that the word the preposition attaches to is more important in predicting the dependent of the preposition. We made this assumption because of the sparsity problem when estimating the full tri-lexical distribution (or quadri-lexical if the preposition is counted). However this independence assumption may be harmful if enough training data is available. From Figure 4.2 and Equation 4.6 it follows we can write the joint model decomposition as follows:

$$\begin{aligned} p(va, V, N_1, P, N_2) &= \\ & P(va)p(P|va)p(V|va, P)p(N_1|va, P, V)p(N_2|va, P, V) \qquad (4.7) \\ p(na, V, N_1, P, N_2) &= \\ & P(na)p(P|na)p(V|na, P)p(N_1|na, P, V)p(N_2|na, P, N_1) \qquad (4.8) \end{aligned}$$

In our final model each of the factors above, except for $P(Att)$ and $p(P|Att)$, is estimated using random walks.

To illustrate the degree of data sparsity for this problem, Figure 4.3 shows the

Figure 4.3: The sparsity of the data: the percent of times tuples in the test set had appeared in the training set.

Factor		**% Non-Zero**
Verbal	$P(p\|va)$	99.8
	$P(v\|va,p)$	64.8
	$P(n_1\|va,p,v)$	15.7
	$P(n_2\|va,p,v)$	13.8
Noun	$P(p\|na)$	99.5
	$P(v\|p,na)$	69.4
	$P(n_1\|na,p,v)$	20.9
	$P(n_2\|na,p,n_1)$	17.4

percentage of test cases for which we had a non-zero relative frequency estimate from the training set for each of the factors needed for Equations 4.7 and 4.8. Here we include cases where the history was seen together with the future at least once. More specifically, we count cases considering their correct class in the test set. For example, of the test cases (v, n_1, p, n_2, Att) with $Att = va$ in the test set, 64.8% had a verb v and preposition p such that p was seen with v in the training set with $Att = va$, i.e., $count(v, p, va) > 0$. These sparsity statistics are collected in a different way compared to both of Dan Bikel's (Bikel, 2004) statistics for the frequency of use of bi-lexical probabilities in Model 2 of Collins (1997) (1.5% and 28.8% of the time, depending on the method of collection). This is because we count frequency with which non-zero lexical probability was obtained for the *correct* analysis, whereas his two methods count the frequency with which the *conditioning context* had a non-zero count across *all* analyses (1.5% of the time) and the *best scoring* analysis (28.8% of the time).

As can be seen, for the factors involving two words in addition to the preposition, more than 3/4 of the time we have not seen the tuple in the training set. The relative frequency estimate $p(N_2|P, V, N_1, Att)$ is non-zero only 5.6% of the time, but we do not model this distribution directly because of our context specific independence assumption described above.

Figure 4.4: Form of factors for BASELINE model. Each factor is estimated as a linear interpolation of the listed empirical distributions.

Factor		Interpolated Distributions	Num Estimated Parameters
Verbal	$P(p\|va)$	$\hat{P}(p\|va)$, $\frac{1}{V_p}$	0
	$P(v\|va,p)$	$\hat{P}(v\|p,va)$, $\hat{P}(v\|va)$, $\hat{P}(v)$, $\frac{1}{V_v}$	$4 \times numBins$
	$P(n_1\|va,p,v)$	$\hat{P}(n_1\|va,p,v)$, $\hat{P}(n_1\|va,p)$,$\hat{P}(n_1\|va)$,$\hat{P}(n_1)$,$\frac{1}{V_{n_1}}$	$5 \times numBins$
	$P(n_2\|va,p,v)$	$\hat{P}(n_2\|va,p,v)$, $\hat{P}(n_2\|va,p)$,$\hat{P}(n_2\|va)$,$\hat{P}(n_2)$,$\frac{1}{V_{n_2}}$	$5 \times numBins$
Noun	$P(p\|na)$	$\hat{P}(p\|na)$, $\frac{1}{V_p}$	0
	$P(v\|na,p)$	$\hat{P}(v\|p,na)$, $\hat{P}(v\|na)$, $\hat{P}(v)$, $\frac{1}{V_v}$	$4 \times numBins$
	$P(n_1\|na,p,v)$	$\hat{P}(n_1\|na,p,v)$, $\hat{P}(n_1\|na,p)$,$\hat{P}(n_1\|na)$,$\hat{P}(n_1)$,$\frac{1}{V_{n_1}}$	$5 \times numBins$
	$P(n_2\|na,p,n_1)$	$\hat{P}(n_2\|na,p,n_1)$, $\hat{P}(n_2\|na,p)$,$\hat{P}(n_2\|na)$,$\hat{P}(n_2)$,$\frac{1}{V_{n_2}}$	$5 \times numBins$

4.5.1 Baseline Model

We describe our baseline method for estimating the conditional probability tables of our model. We refer to it as BASELINE. We will see later that it is a special case of our more general random walk models. Our baseline model can be seen as a linearly interpolated model of relative frequency estimates as in Jelinek-Mercer smoothing (Jelinek and Mercer, 1980), where the interpolation weights are fit on held-out data.

For example, the probability $P(n_2|va,p,v)$ is estimated as follows:

$$\begin{aligned} P(n_2|va,p,v) &= \lambda_0(va,p,v)\hat{P}(n_2|va,p,v) \\ &+ \lambda_1(va,p,v)\hat{P}(n_2|va,p) \\ &+ \lambda_2(va,p,v)\hat{P}(n_2|va) \\ &+ \lambda_3(va,p,v)\hat{P}(n_2) + \lambda_4(va,p,v)\frac{1}{V_{n_2}} \end{aligned} \tag{4.9}$$

The other factors of the model are similarly estimated using interpolation of varying order distributions. Figure 4.4 lists the interpolation order for all factors. The figure also shows the number of tunable interpolation parameters for each factor. The $\hat{P}$ distributions are relative frequency estimates from the training data. In Jelinek-Mercer smoothing, the interpolation weight for a distribution depends on the binned count of the conditioning context as described in §4.3, Equation 4.3. We also estimate parameters based on binned counts, but we bin only the most specific contexts, for

example, (va, p, v) in Equation 4.9, and the multiplier for each of the interpolated distributions depends only on this bin. There is no difficulty in incorporating binned counts for lower order contexts as well.

In Figure 4.4, $numBins$ in the third column stands for the number of bins we have chosen for the interpolation parameters. The number of estimated parameters for our model thus depends on the chosen number of bins. The number of estimated parameters for $P(p|Att)$ is shown as 0. This is because, since this distribution is not sparse, we used a simpler interpolation method that does not require parameter fitting and estimated the interpolation weights as closed-form expressions depending on the counts of the context (Witten-Bell smoothing, (Witten and Bell, 1991)). For a given context, the interpolation weights are constrained to sum to 1. Therefore the number of free parameters is one less than shown for each bin. If there is a single interpolation parameter for each of the terms shown in each of the rows, i.e. $numBins = 1$, the number of free trainable parameters of the model will be 22.

As in standard held-out Jelinek-Mercer interpolation, we fit the interpolation parameters to maximize the likelihood of a held-out set of samples. If we are maximizing joint log-likelihood, we could use EM for optimization. However, since our final task here is making a classification decision, we could be able to profit from discriminative estimation (Klein and Manning, 2002; Ng and Jordan, 2002; Toutanova et al., 2003).

To test this hypothesis, we experimented with maximizing the conditional log-likelihood of the held-out set instead. To maximize conditional log-likelihood with respect to the interpolation parameters, we need more general optimization algorithms than EM. To avoid constrained optimization, we handled the constraints $\sum_i \lambda_i(context) = 1, \forall context$ and $\lambda_i(context) \geq 0$ by representing $\lambda_i(context) = e^{\gamma_i(context)} / \sum_{i'} e^{\gamma_{i'}(context)}$. The new model parameters are the $\gamma_i(context)$ and they are not constrained. Here $context$ represents the conditioning context on which an interpolation parameter depends, for example (va, p, v) for estimating $P(n_2|va, p, v)$, and i' ranges over the different levels of back-off. We also add a gaussian prior on the new parameters γ. The regularized joint and conditional log-likelihood functions are

as follows:

$$JL: \quad \sum_{i=1,\ldots,N} \log P(Att^i, v^i, {n_1}^i, p^i, {n_2}^i) - \frac{1}{2\sigma^2} \sum_{s=1,\ldots,k} {\gamma_s}^2 \tag{4.10}$$

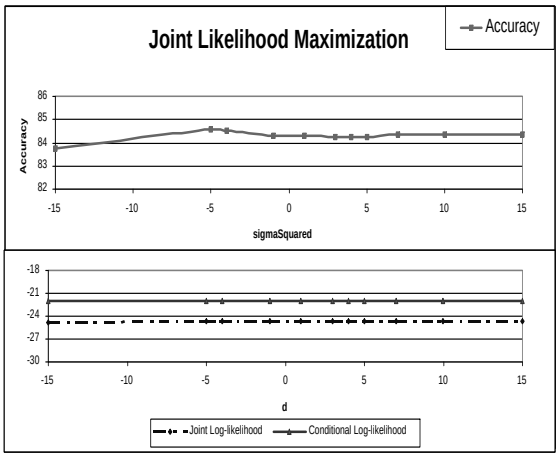
$$CL: \quad \sum_{i=1,\ldots,N} \log P(Att^i | v^i, {n_1}^i, p^i, {n_2}^i) - \frac{1}{2\sigma^2} \sum_{s=1,\ldots,k} {\gamma_s}^2 \tag{4.11}$$

Here i ranges over the held-out set of samples and s ranges over model parameters. The number k of model parameters depends on the number of equivalence classes (bins). If the number of bins is one, k is 28.

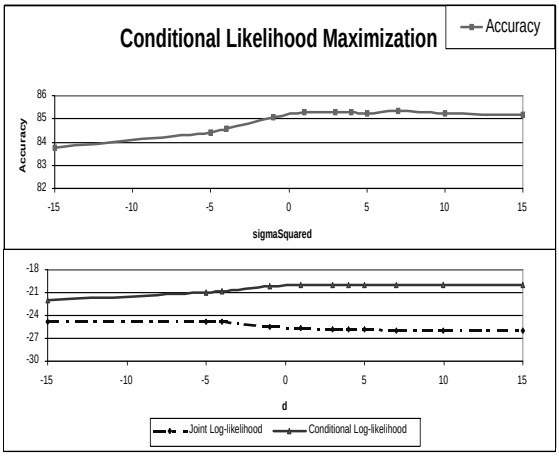
4.5.2 Baseline Results

For comparison with previous work, we train on the training set described in §4.4.1, fit interpolation parameters on the development set and report results on the test set. To avoid overfitting on the test set, we first performed experiments without evaluating on the test set. To do this, we took the first 17,801 examples from the training set as a preliminary training set and the remaining 3,000 examples as a preliminary test set - called PreTest. The development set of 4,039 was used as a held-out set for fitting interpolation parameters in these preliminary experiments as well. Limited memory Quasi-Newton was used for numerical optimization.

We first study the performance of the Baseline model when trained on the reduced training set and tested on PreTest. We compare the performance of the model when trained to maximize the regularized joint log-likelihood 4.10 versus the regularized conditional log-likelihood 4.11, for varying values of the smoothing parameter *sigmaSquared* of the Gaussian prior. A single equivalence class is used for all interpolation parameters. We found that increasing the number of equivalence classes did not result in significant improvement in accuracy. Figure 4.5 shows the results of this experiment; the results for joint log-likelihood training are on the left in 4.5(a), and the results for conditional log-likelihood are on the right in 4.5(b). The horizontal axis shows *sigmaSquared* on a logarithmic scale, varying between 2^{-15} and 2^{15}. The joint log-likelihood is shown as an average per test sample, and the conditional log-likelihood is shown scaled by the constant of $\frac{1}{50}$ to make it more easily displayable on



(a) Joint likelihood maximization.



(b) Conditional likelihood maximization.

Figure 4.5: Performance of the BASELINE model when trained to maximize the joint or the conditional likelihood of the development set. Accuracy, average joint, and scaled conditional log-likelihood on test set PreTest are shown for varying values of the regularization parameter *sigmaSquared*.

the same graph.

We can see that the maximum accuracy achieved by the model trained for conditional log-likelihood is higher than the maximum achieved by the model trained for joint log-likelihood (85.33% versus 84.57%). For reference, the accuracy of the Collins & Brooks model on PreTest is 83.37%. The behavior of the curves is intuitive – the model trained for conditional log-likelihood achieved better conditional log-likelihood and accuracy, and worse joint log-likelihood, as compared to the model trained for joint log-likelihood. When the smoothing parameter *sigmaSquared* is very small, the weight of the prior dominates and the model is forced to set all γ parameters near zero. Then both models are nearly uniform and have the same performance. When the variance *sigmaSquared* is increased, the performance improves for a while until the prior becomes too weak and the model starts to over-fit slightly.

Figure 4.6 compares the accuracy of our baseline model to other previously reported results on the same data set - the IBM dataset described in §4.4.1. The first

result reported in the figure is from the Collins & Brooks algorithm which we reviewed in §4.4.2. This is the result exactly as reported in the paper (Collins and Brooks, 1995), and line two shows the result of our re-implementation on the preprocessed dataset. The preprocessing was described in §4.4.1. The model in line three is our implementation of the C&B model run on a version of the data further processed to substitute all verbs with their root forms according to the morphological analyzer developed by Kevin Humphreys , John Carroll, and Guido Minnen (Minnen et al., 2001). Such an experiment was also reported in (Collins and Brooks, 1995). In lines four, five, and six, we list previously reported results on the same dataset for other machine learning algorithms, which we reviewed in §4.4.2.

Line seven shows the performance of BASELINE when trained to maximize the regularized joint log-likelihood with value of $sigmaSquared$=2^{-5}, its optimal value according to the PreTest experiment. Line eight is BASELINE trained to maximize conditional log-likelihood with $sigmaSquared = 128$ (the optimal value according to the PreTest experiment.)

It is worth noting that our simple generative model with linearly interpolated relative frequency estimates (and interpolation parameters fit discriminatively), performs significantly better than the discriminative back-off C&B algorithm. Statistical significance is measured using McNemar's test (Everitt, 1977; Dietterich, 1998). The model BASELINE trained for regularized conditional log-likelihood outperforms the C&B algorithm with significance $< .005$. The model trained for regularized joint log-likelihood outperforms it at a lower significance level. Results of the statistical significance tests are shown in the last column of Figure 4.6.

4.6 Random Walks for PP Attachment

We now describe our random walk model for the word dependency distributions needed for equations 4.7–4.8. We illustrate with the case of estimating $p(N_2|P, V, va)$. Instantiating the example in (3), this is $P(N_2 = peg|P = with, V = hang, va)$, the probability that, given *hang* is modified by a PP whose head is *with*, *peg* is the head

Figure 4.6: BASELINE results on the final test set of 3,097 samples, compared to previous work. In the significance column > means at level .05 and ≫ means at level .005.

NUMBER	MODEL	ACCURACY	SIGNIF
1	C&B (COLLINS AND BROOKS, 1995)	84.15%	
2	C&B O.R.	84.18%	
3	C&B STEM VERBS O.R.	84.50%	NOT > 2
4	K-NN (JAKUB ZAVREL AND VEENSTRA, 1997)	84.40%	
5	BOOSTING (STEVEN ABNEY, 1999)	84.40%	
6	SVM (VANSCHOENWINKEL AND MANDERICK, 2003)	84.80%	
7	BASELINE JL	85.37%	> 2
8	BASELINE CL	85.89%	NOT > 7,≫ 2
9	BASELINE CL STEM VERBS	86.05%	NOT > 7,≫ 3

of the noun phrase governed by *with*. This is strictly a tri-lexical dependency, but because prepositions can often be regarded as just a marker of the semantic role of their object noun phrase, we can informally think of this as estimating the probability of a particular sort of semantic dependency; here it is the likelihood of n_2:peg bearing a *with*-type dependency to the word v:hang. Thus, given the preposition, we can view this as estimating a bi-lexical dependency between a verb v and a noun n_2.

We will estimate this probability using a Markov Chain. More precisely, we will construct a MC M (whose transition probabilities will depend on p, v, and the fact that $Att = va$) so that its stationary distribution π is a good approximation to $P(n_2|p, v, va)$.

We let the state space $\mathcal{S}$ of our random walk be $\mathcal{W} \times \{0, 1\}$, where $\mathcal{W}$ is the set of all words. Thus, a state is a pair consisting of a word and a single "bit" taking on a value of 0 or 1. As we will shortly see, the extra memory bit allows our walk to "remember" if the word in the current state is a head (0) or a dependent (1), and will permit us to build richer models.[2] For $P(n_2|p, v, va)$, v is a head, and n_2 is a dependent (and the type of the dependency relationship is indicated by p). Below we will write $(w, 0)$ as ${}^h w$ and $(w, 1)$ as ${}^d w$, both for brevity, and to remind us of the

[2]Other examples of Markov Chains that can be thought of as random walks with an extra memory bit include (Lafferty and Zhai, 2001; Ng et al., 2001).

extra bit's meaning.

The initial distribution p_0 of our Markov Chain puts probability 1 on the state ${}^h v$ (i.e., we always start at the state for the head verb, with the bit-value 0).

Let us first walk through some cases using the "hang painting with peg" example, with the small random walk model shown in Figure 4.7. We are trying to estimate

$$P(N_2 = peg|V = hang, P = with, Att = va). \tag{4.12}$$

If, in a training set of disambiguated PP-attachment examples, we have seen the event $(V = hang, P = with, Att = va)$ before, then clearly one possible estimate for the probability in (4.12) might be given by its empirical distribution. Specifically, if *peg* was frequently seen in the context of the event $(V = hang, P = with, Att = va)$, then we would like to assign a large probability to this event. One way to ensure that the random walk frequently visits *peg* in this setting is therefore to have the probability of transitioning from the initial state to some other state ${}^d w$, representing a dependent word, be monotonically increasing in the empirical distribution of $P(N_2 = w|V = hang, P = with, Att = va)$.

Now, suppose that, because of data sparseness problems, we have not seen "v:hang p:with n_2:peg" in our training set, but that we have seen "v:hang p:with n_2:pegs" several times. Further, our stemmer indicates that *peg* and *pegs* have the same root form. In this setting, we would still like to be able to assign a high probability to $P(peg|hang, with, va)$. I.e., we want π to give ${}^d peg$ a large probability. Using the state transitions described above, we already have a large probability of visiting ${}^d pegs$. If our random walk now gives a large probability of transitioning from ${}^d pegs$ to ${}^d peg$, then we would be done. More broadly, we would like our random walk to be able to make a transition from (w_1, b_1) to (w_2, b_2), if w_1 and w_2 are words with the same root form, and $b_1 = b_2$.

Similarly, if we know that $P(rivet|hang, with, va)$ has a large probability, and if some external knowledge source tells us that *rivet* and *peg* are semantically closely related, then we should infer that $p(peg|hang, with, va)$ should also be fairly large. This can be done by using a thesaurus, or a resource like WordNet, a large collection

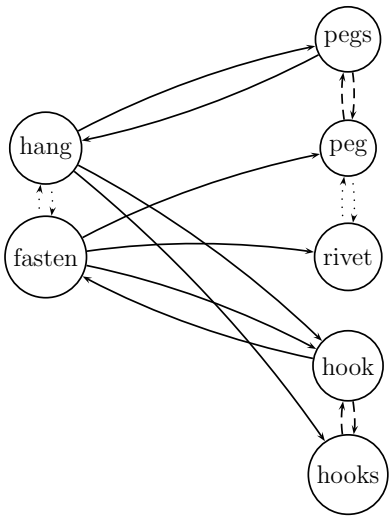


Figure 4.7: A small words state space for learning the distribution $P_{with}(n_2|v)$.

of words classified into a set of senses (synsets), which are organized in a hierarchy, and permitting transitions between (w_1, b_1) and (w_2, b_2) if an external knowledge source tells us that w_1 and w_2 are related, and $b_1 = b_2$.[3]

More broadly, we have outlined above several different "types" of inferences that can be made about what tuples v, p, n_2 are likely. These types of inferences often exploit external knowledge sources (such as a stemmer, or WordNet), and we have shown several examples of how they can be encoded into a random walk framework, so that the stationary distribution gives a large probability to events that we would like our procedure to conclude are likely. Note in particular that if there are multiple paths to a node, then that "reinforces" a particular conclusion. By combining multiple steps of these inferences together, in Figure 4.7, we should be able to conclude that if (a) *hang with hooks*, (b) *fasten with hook* and (c) *fasten with rivet* are likely; that (d) *hooks* and *hook* have the same root, and if (e) *rivet* and *peg* are semantically related, then *hang with peg* is also likely. Specifically, the sequence of states the random walk

[3]Of course, some of these could lead to incorrect inferences—even though *hang with peg* may be likely, and WordNet indicates that peg and nail-polish are semantically related (via nail), it is incorrect to infer that *hang with nail-polish* is therefore likely. However, we will later describe how a learning algorithm is used to automatically decide the degree to which each of these inferences can be trusted.

might visit based on this information is ${}^{h}hang \xrightarrow{a} {}^{d}hooks \xrightarrow{d} {}^{d}hook \xrightarrow{b} {}^{h}fasten \xrightarrow{c} {}^{d}rivet \xrightarrow{e} {}^{d}peg$. Thus, by considering multiple steps of the random walk, we can combine multiple steps of inference together. But the model by its nature also captures that long multi-step chains do not give much support to their conclusion.

4.6.1 Formal Model

We now describe our model formally. The reader may wish to review the Markov Chain preliminaries of §4.2 at this point. We are using a Markov Chain model to estimate a distribution $p({}^{d}W|{}^{h}W)$ over dependent words given head words for a particular dependency relationship. The state space $\mathcal{S}$ of our random walk is $\mathcal{W} \times \{0, 1\}$, where $\mathcal{W}$ is the set of all words, 0 indicates a head, and 1 a dependent state. Since we need a conditional distribution over dependent states given a head state, we define a separate Markov Chain for each head word. All Markov Chains that estimate the same type of dependency relationship use a common transition matrix P as a building block. The Markov Chain for estimating a conditional distribution $p({}^{d}W|{}^{h}w_*)$ for a fixed head word ${}^{h}w_*$ is defined using an initial distribution $p_{0,{}^{h}w_*}$ and a state transition distribution $p_{{}^{h}w_*}(w', b'|w, b)$ defined as follows:

$$p_{0,{}^{h}w_*}(w, b) = 1 \iff (w, b) = (w_*, 0)$$

$$p_{{}^{h}w_*}(w', b'|w, b) = \gamma({}^{h}w_*)p_{0,{}^{h}w_*}(w', b') + (1 - \gamma({}^{h}w_*))P(w', b'|w, b)$$

In other words, the initial distribution places probability 1 on the given head word, and the state transition distribution is obtained by interpolating the initial distribution with the distribution P, which is common for all head words. Here, slightly abusing notation, P is used both as a distribution and as a matrix. The Markov Chain for a given head word ${}^{h}w_*$ is thus of the same form as the familiar Markov Chains with a positive probability of resetting according to the initial distribution at each state (Equation 4.1 in §4.2). As discussed in §4.2, the stationary distribution of a Markov Chain of this form is the same as the distribution of states generated according to the following generative process: start at state S_0 distributed according

to $p_{0,{}^h w_*}$ (i.e., start at ${}^h w_*$); at each following step $t+1$, with probability $\gamma({}^h w_*)$, stop the process and output the previous state S_t, and with probability $1-\gamma({}^h w_*)$, take a step according to the transition distribution P.

We are interested in estimating the transition matrix P. We define P using a set of different **links**, which should be thought of as "basic" transition distributions that correspond to different possible inference steps. Each link type always leads from states where the memory bit takes on some particular value b_1 to states where the bit takes on a value b_2 (not necessarily different from b_1). The final transition distribution P will then be a mixture of the basic transition distributions, where the mixture weights are learned automatically.

Let the **links** $l_1, \ldots, l_k$ be given by transition matrices $P_1, \ldots, P_k$. Each matrix P_i has rows for states with memory bits $startBit(i)$ and its rows are distributions over successor states with memory bit $endBit(i)$. For example, a morphology link might be given by a transition matrix that specifies a uniform transition distribution from dependent nouns to other dependent nouns with the same root form. The probability of transitioning from (w, b) to (w', b') according to the estimated P is given by:

$$P(w', b'|w, b) = \sum_{i:startBit(i)=b, endBit(i)=b'} \lambda_i(w, b) P_i(w', b'|w, b)$$

The parameter $\lambda_i(w, b)$ is the weight of link l_i for the state (w, b). It can also be viewed as the probability of taking a link of type l_i given the current state (w, b). The probabilities $\lambda_i(w, b)$ sum to 1 over all links l_i having a starting bit $startBit(i) = b$. Parameters of this form for all states are estimated automatically from data. Since estimating separate parameters for each word would introduce too much sparsity, we define equivalence classes of states for which we tie the parameters. Notice the similarity to Jelinek-Mercer smoothing and our BASELINE model, described in §4.5.1.

As mentioned in §4.2, we also add one more refinement to the model, by further distinguishing between two different kinds of links: ones that can be followed at any time, and ones that can be taken only in a final step of the walk (before the generative process is stopped). We call the latter type **final** links. The intuition here

is that (due to the usual sparseness in NLP data) we do wish to include in our model distributions that back off from conditioning on individual words and that therefore can transition according to a highly-smoothed model. But, it would be undesirable to allow transitions to backed-off distributions throughout the random walk. Specifically, allowing such transitions would cause us to lose the intuition of the random walk as exploring close neighbors of a word based on some similarity criterion. An additional advantage of having a special stopping distribution is that we can disable transitions to states that don't have the desired memory bit; for example, because we want the random walk to estimate $p({}^{d}W|{}^{h}w_{*})$, the last state has to be a dependent. Thus in a final step of the walk, we fix the the probability of following a link type leading to a non-dependent state to zero.

Thus we learn two different transition distributions for the Markov Chains — a distribution $P_{nfin}(w', b'|w, b)$, and a distribution $P_{fin}(w', b'|w, b)$. The **final** links participate only in P_{fin}, whereas the other links participate in both P_{fin} and P_{nfin}.

The stationary distribution for a Markov Chain defined in this way, that estimates the probability distribution over dependent states given a head word state ${}^{h}w_{*}$, will be:

$$\pi = \gamma({}^{h}w_{*}) \sum_{t=0}^{\infty} (1 - \gamma({}^{h}w_{*}))^{t} p_{0,{}^{h}w_{*}} (P_{nfin})^{t} P_{fin}$$

This follows from the discussion in §4.2 and corresponds to Equation 4.2. For computational reasons, we limit the maximum number of time steps to some number d; we call d the maximum degree of the Markov chain. In this case the equation above is truncated to $t = d - 1$ terms and re-normalized to sum to 1.[4] Additionally, one might think that the geometric distribution of the number of steps is too restrictive. Especially when we have two separate types of states – head and dependent states, there may be some inferences that are only possible at odd or even number of steps and the geometric distribution may be inappropriate. Therefore, in the most general form of the model we allow for arbitrary weighting factors for each time step t. The form of the estimated distribution over dependents given a head state ${}^{h}w_{*}$, using a

[4]The limit is $d - 1$ and not d because the final transition step according to P_{nfin} also counts toward the number of steps.

maximum degree d is defined as follows:

$$\tilde{p}({}^{h}w_{*}) = \sum_{t=0}^{d-1} \gamma_t({}^{h}w_{*}) p_{0,{}^{h}w_{*}} (P_{nfin})^t P_{fin} \tag{4.13}$$

The parameters of the model are the interpolation parameters in the final and non-final transition matrices P_{fin} and P_{nfin}, as well as the weighting factors of the number of steps t – $\gamma_t({}^{h}w_{*})$.

PP Attachment Dependency Distributions

We have been describing a Markov Chain for a specific type of dependency distribution $p({}^{d}W|{}^{h}W)$. This construction is directly applied to estimating the factors needed for our generative PP attachment model, illustrated in Figure 4.2. The conditional probability tables for the random variables V, N_1, and N_2 are estimated using Markov chains. The conditional distribution for V is $p(V|Att, P)$. For this dependency distribution, the prepositions are heads, the verbs are dependents, and the type of dependency relation is determined by Att. When $Att = va$, the preposition modifies the verb, and when $Att = na$, the preposition does not modify the verb but modifies a noun phrase following the verb. We learn two separate pairs of transition matrices ,$(P_{va,fin},\ P_{va,nfin})$ and $(P_{na,fin},\ P_{na,nfin})$, for the two types of dependency relationship. For the conditional distribution at N_2, we need to estimate $p(N_2|va, P, V)$ and $p(N_2|na, P, N_1)$. These distributions are defined similarly to the conditional distributions at N_2 – $p(N_1|va, P, V)$ and $p(N_1|na, P, V)$. A separate pair of finite and non-finite transition matrices is learned for each pair Att, P in the conditioning context of the distributions over N_1 and N_2. For example, as informally described in the beginning of §4.6, for the distribution over second nouns, we learn transition matrices $(P_{att,p,fin},\ P_{att,p,nfin})$, for each preposition p and each attachment type att.

4.6.2 Parameter Estimation

The parameters of the model were fit to optimize the conditional log-likelihood of the correct attachment sites for a development set of samples, disjoint from the training and test sets, including quadratic regularization. That is, as in the BASELINE model, we maximized the objective:

$$\sum_{i=1,\ldots,N} \log P(Att^i | v^i, {n_1}^i, p^i, {n_2}^i) - \frac{1}{2\sigma^2} \sum_{s=1,\ldots,k} {\theta_s}^2$$

Here i ranges over the sample set, and s ranges over the model parameters. We performed the optimization using a limited memory quasi-Newton method. Here again as in BASELINE, we use the logistic transformation to avoid constrained optimization. We represent the probabilistic λ and γ parameters with real-valued unconstrained θ parameters, as will be made precise shortly.

The number of parameters depends on the scheme for defining equivalence classes over states. The parameters correspond to distributions over link types for states and probabilities of number of time steps t. We experimented with binning the parameters based on observed number of times of occurrence of words. For the link type parameters in P_{fin} and P_{nfin}, the simplest model, having a single equivalence class, performed on average as well as the more complex models. This scheme of binning the parameters also ties the parameters of corresponding link types for the matrices corresponding to different prepositions (but not different attachment types). Section 4.6.5 illustrates the parameters for a specific model.

For the parameters for distributions over number of time steps t, we found that binning the parameters in two bins was consistently helpful. We found that it was advantageous to group the time step parameters not only based on the head word, but also on the dependent word – i.e., based on $count({}^h w, {}^d w)$. The equivalence classing that helped was to have a class of pairs of head and dependent words whose count was 0 and another class for pairs whose count was greater than 0. Note that because of this binning scheme the scores from the random walk model can no longer be interpreted as probabilities, because they do not sum to one over dependent words.

A binning scheme which caused a similar lack of probabilistic interpretation was used in the parser of Charniak (Charniak, 2000). This is not a problem for our model because we use it as a conditional model over attachment sites given the four head words and it does sum to 1 over the two possible values of the attachment. This binning scheme was only slightly but consistently better than a single equivalence class over all words.

To estimate the objective function using the quasi-Newton method, we need to be able to compute its value and gradient for a given parameter setting Θ. We can compute the value of the conditional word distributions estimated using Markov Chains in time linear in the maximum degree d of the walk, and linear in the size of the state space. The algorithm for computation of derivatives that we use is quadratic in d and linear in the size of the state space.

To describe the algorithms for computing the value and gradient of the objective function, we recap the notation we are going to use. We concentrate on a single Markov Chain model for estimating a dependency distribution such as $\tilde{p}({}^{d}w|{}^{h}w)$ of dependent words given head words. Having done that, it is easy to combine the values and gradients of the different factors in the conditional log-likelihood, using the rules of differentiation of composite functions.

We will denote by Θ a vector of parameters – real-valued weights. We denote by $\mathcal{S}$ the state space of the Markov Chain model, $\mathcal{S} = \mathcal{W} \times \{0, 1\}$. Each state s is a pair of a word and a bit indicating head or dependent state and $bit(s)$ is the bit of state s.

The Markov Chain models are built up of given link matrices $P_{i,nfin}$ and $P_{i,fin}$. Let $\lambda_{i,nfin}(s)$ denote the probability of taking link type i at a non-final transition step. As discussed above, the non-negative normalized parameters $\lambda_{i,nfin}$ are defined via real-valued parameters $\theta_{i,nfin}$ as follows:

$$\lambda_{i,nfin}(s) = \frac{e^{\theta_{i,nfin}(s)}}{\sum_{i':startBit(i')=bit(s)} e^{\theta_{i',nfin}(s)}}$$

The representation of the $\lambda_{i,fin}(s)$ is defined correspondingly. A maximum degree of the walk d is specified. The probabilities over the number of non-final time steps

$time = 0, \ldots, d-1$ are $\gamma_{time}(s)$ for a starting head state s. These parameters are also defined via real-valued θ parameters.

$$\gamma_{time}(s) = \frac{e^{\theta_{time}(s)}}{\sum_{time'=0}^{d-1} e^{\theta_{time'}(s)}}$$

The non-final and final transition matrices depending on Θ are written as $P_{\Theta,nfin}$ and $P_{\Theta,fin}$. We will suppress the dependence on Θ when it is clear from the context. We are interested in computing the value and gradient of $\tilde{p}_\Theta(end|begin)$, where *begin* and *end* are a head and dependent state observed in the held-out data to occur in the dependency relationship. Let $\vec{b'}$ denote a row vector which represents the initial probability distribution of the Markov Chain starting at the state *begin* – i.e., $\vec{b'}$ places probability 1 on *begin* and 0 on all other states. Let $\vec{e}_\Theta$ denote the column vector with entries $\vec{e}_\Theta(s) = P_{\Theta,fin}(end|s)$ for all states $s \in \mathcal{S}$. Then the value of the estimated dependency distribution is:

$$\tilde{P}_\Theta(end|begin) = \sum_{time=0,\ldots,d-1} \gamma_{time}(begin)\vec{b'}P_{\Theta,nfin}^{time}\vec{e}_\Theta$$

The algorithm for computing the value of an estimated distribution according to Equation 4.13 is as follows:

1. Initialize $\vec{c'} = \vec{b'}$, $\tilde{P}(end|begin) = \gamma_0(begin)\vec{c'}$, $time = 1$
2. If $time = d$, goto 5.
3. $\vec{c'} = \vec{c'}P_{\Theta,nfin}$, $\tilde{P}(end|begin) = \tilde{P}(end|begin) + \gamma_{time}\vec{c'}$, $time = time + 1$.
4. goto 2.
5. $\tilde{P}(end|begin) = \tilde{P}(end|begin)\vec{e}_\Theta$

As we can see, at each step three we need to multiply a vector with the non-final transition matrix, multiply the vector by a number and add two vectors; then in step five we perform a dot product. Therefore the estimated value can be computed in time $O(d(F_+|\mathcal{S}_{d-1}|))$, where F_+ is the number of non-zero entries in the non-final

transition matrix (which is very sparse in our application) and $|\mathcal{S}_{d-1}|$ is the size of the state space reachable within $d-1$ transitions according to the non-finite transition matrix from the start state *begin*.

The algorithm that we use for computing the gradient is similar to the Baum-Welch (forward-backward) algorithm for computing derivatives for Hidden Markov Models (Baum, 1972). More efficient algorithms may exist. For example, (Eisner, 2001) describes optimizations for a similar model.

To compute the derivative of an observed event $\tilde{P_\Theta}(end|begin)$, we need to compute expectations over number of times a particular transition has occurred. The hidden variables are the number of time steps *time* before the Markov chain has stopped, the link types that have been followed at each step and the intermediate states visited between *begin* and *end*. To compute the gradient with respect to one of the unnormalized real-valued parameters θ_i, we can first take gradients with respect to all λ'_i (or γ'_i) parameters which depend on θ_i. For example, if $\lambda_i = \frac{e^{\theta_i}}{\sum_{i'} e^{\theta'_i}}$, then:

$$\frac{\partial \tilde{P_\Theta}(end|begin)}{\partial \theta_i} = \sum_{i'} \frac{\partial \tilde{P_\Theta}(end|begin)}{\partial \lambda'_i} \frac{\partial \lambda'_i}{\partial \theta_i}$$

It is straightforward to differentiate with respect to the $\gamma_{time}(begin)$ parameters because $\tilde{P_\Theta}(end|begin)$ is a linear function of these. The derivative with respect to a particular $\gamma_{time}(begin)$ is the following:

$$\frac{\partial \tilde{P_\Theta}(end|begin)}{\partial \gamma_{time}(begin)} = \vec{b}' P^{time}_{\Theta,nfin} \vec{e}_\Theta$$

which is proportional to one of the terms needed for estimating the value of $\tilde{P_\Theta}(end|begin)$. Differentiating with respect to the $\lambda_{i,fin}(s)$ parameters is similarly straightforward, because a final transition is taken only once and therefore $\tilde{P_\Theta}(end|begin)$ is a linear expression of such parameters. We will give the expression of these derivatives after introducing some notation in connection with the $\lambda_{i,nfin}(s)$ derivatives.

Differentiating with respect to the $\lambda_{i,nfin}(s)$ parameters is trickier because the non-final transition matrix is raised to powers between 0 and $d-1$. For these derivatives,

we apply a forward-backward style dynamic programming algorithm to compute the probability that a given link type i has been taken from state s at time *time*. The derivative with respect to the parameter $\lambda_{i,nfin}(s)$ is proportional to the expected number of times non-final link type i is followed starting from s. More specifically, if the probability that we take link type $L_q = l_{i,nfin}$ out of state $S_q = s$ at a non-final step q according to our Markov Chain model starting at *begin* and ending at *end* is denoted by $\tilde{P}_\Theta(end, S_q = s, L_q = l_{i,nfin}|begin)$, the derivative is:

$$\frac{\partial \tilde{P}_\Theta(end|begin)}{\partial \lambda_{i,nfin}(s)} = \frac{1}{\lambda_{i,nfin}(s)} \sum_{time=0}^{d-2} \tilde{P}_\Theta(end, S_{time} = s, L_{time} = l_{i,nfin}|begin)$$

.

To compute the necessary quantities, we will use forward and backward factors. We define the forward probability $\alpha^{time}(s)$, $time = 0, \ldots d-1$ as the probability of reaching state s in exactly *time* transition steps according to the non-final transition distribution.

$$\alpha^{time} \stackrel{def}{=} \vec{b}' P^{time}_{\Theta,nfin}$$

Evidently the forward probabilities can be computed similarly to the algorithm for computing the value of $\tilde{P}_\Theta(end|begin)$, in time $O(dF)$. We need space $O(d|\mathcal{S}_{d-1}|)$ to store the d forward row vectors. Having computed the forward probabilities we can compute the derivatives with respect to the final interpolation parameters as follows:

$$\frac{\partial \tilde{P}_\Theta(end|begin)}{\partial \lambda_{i,fin}(s)} = (\sum_{time=0}^{d-1} \gamma_{time}(begin)\alpha^{time}(s))P_{i,fin}(end|s)$$

The backward probability $\beta^{remtime}(s)$ for $remtime = 1, \ldots, d$ is defined as the probability of ending at state *end* after *remtime* time steps, taking $remtime - 1$ non-final transitions and one final transition.

$$\beta^{remtime} \stackrel{def}{=} P^{remtime-1}_{\Theta,nfin} \vec{e}_\Theta$$

We also define backward factors specific to non-final link types $l_{i,nfin}$. We define ${\beta_{i,nfin}}^{remtime}(s)$, for $remtime = 2, \ldots, d$, as the probability of reaching *end* from s via taking the non-final link $l_{i,nfin}$ first, then taking $remtime - 2$ additional non-final transitions and then one final transition.

$$ {\beta_{i,nfin}}^{remtime}(s) \stackrel{def}{=} \lambda_{i,nfin}(s) \vec{b_s}' P_{i,nfin} P_{\Theta,nfin}^{remtime-2} \vec{e}_\Theta $$

Here, the row vector $\vec{b_s}'$ places probability 1 on s and 0 on all other states. Evidently, we can calculate the β and $\beta_{i,nfin}$ factors recursively, starting from $remtime = 1$ via the algorithm listed below. We use $\lambda_{i,nfin}$ to denote the column vector of the parameters for non-final link i for all states s. The total number of non-final links is m. The transition matrix for non-final link type i is $P_{i,nfin}$. $x \bullet y$ stands for the coordinatewise product of two vectors – a vector whose coordinates are products of the respective coordinates of x and y.

1. $remtime = 1$, $\beta^{remtime} = \vec{e}_\Theta$, $remtime = remtime + 1$
2. If $remtime = d + 1$ goto 6
3. for $(i = 1, \ldots, m)$ { ${\beta_{i,nfin}}^{remtime} = \lambda_{i,nfin} \bullet (P_{i,nfin} \beta^{remtime-1})$ }
4. $\beta^{remtime} = \sum_{i=1}^{m} {\beta_{i,nfin}}^{remtime}$
5. $remtime = remtime + 1$, goto 2
6. end

The space required for the $\beta^{remtime}$ and ${\beta_{i,nfin}}^{remtime}$ is $O(d(m+1)|\mathcal{S}_{d-1}|)$. The time required by the algorithm is $O(d(F_1 + F_2 + \cdots + F_m + m|\mathcal{S}_{d-1}|))$, where F_i is the size of matrix $P_{i,nfin}$. This is because in the loop in step three we are doing at most F_i computations for each of the link types and in step four we are summing over m vectors of size at most $|\mathcal{S}_{d-1}|$.

Now having introduced the α and β factors, we are ready to compute the derivatives with respect to the non-final interpolation parameters:

$$\frac{\partial \tilde{P}_\Theta(end|begin)}{\partial \lambda_{i,nfin}(s)} = \frac{1}{\lambda_{i,nfin}} \sum_{time=1}^{d-1} \gamma_{time}(begin) \sum_{q=0}^{time-1} \alpha^q(s) {\beta_{i,nfin}}^{time-q+1}(s)$$

If we implement this equation directly, we have an algorithm quadratic in $d-1$, which we need to apply for every state $s \in \mathcal{S}_{d-1}$ and for every link type. If the γ_{time} parameters factored as $\gamma_{time}(begin) = \gamma_q(begin)\gamma_{time-q}(begin)$ (which would be the case if we had an un-normalized geometric distribution over time steps), we would be able to make the algorithm linear in d, rather than quadratic. However, here we chose to fit a general multinomial distribution over time steps and hence could not apply this optimization. With only a few equivalence classes, further savings in time and memory requirements are possible.

4.6.3 Link Types for PP Attachment

The particular links for PP attachment are different for every type of dependency relationship needed for the generative model. The Markov Chains for computing the conditional distributions over the N_1 and N_2 random variables are more complex than the Markov Chains for computing the distributions at V. We will define the link types in the context of the $p(N_2|va, P, V)$ distributions. There is a direct correspondence between these link types and the ones used for any $p(N|Att, P, H)$ distribution, where N is N_1 or N_2, H is V or N_1 and Att is va or na. For the Markov Chains computing $p(V|Att, P)$, the used link types correspond to a subset of the ones used for the distributions over N_1 and N_2. This is because we do not wish to use similarity measures among prepositions in the same way we use similarity measures among nouns or verbs. In §4.6.5, we list the link types for all dependency relationships in a concrete model.

For modeling $P(N_2|p, v, va)$, we have a separate pair of a final and non-final Markov Chain transition matrix for each preposition p, with the link types given below. The initial state distribution places probability 1 on the state ${}^h v$. The first eight link types are:

1. **V → N.** Transitions from ${}^{h}w_1$ to ${}^{d}w_2$ with probability proportional to the empirical probability of $p(N_2 = w_2|V = w_1, p, va)$. This transition probability is smoothed with Lidstone smoothing (Lidstone, 1920), $\alpha = .01$. (**L1**)

2. **Morphology.** Transitions from (w_1, b) to (w_2, b) for all words w_2 that have the same root form as w_1, with probability proportional to the empirical count of w_2 plus a small smoothing parameter $\alpha = .1$ (Lidstone smoothing). Self-transitions are included as well. (**L2Nouns, L2Verbs**)

3. **WordNet Synsets.** Transitions from states (w_1, b) to (w_2, b), for all words w_2 in the same WordNet synonym-set as one of the top three most common senses of w_1, with probability proportional to the empirical count of w_2 plus a small smoothing parameter $\alpha = .1$. Each morphological form of each word in the top three senses is included. (**L3Nouns, L3Verbs**)

4. **N → V.** Transitions from ${}^{d}w_1$ to ${}^{h}w_2$ with probability proportional to the empirical probability of $p(V = w_2|N_2 = w_1, p, va)$. Like link type L1, it is smoothed using Lidstone smoothing, $\alpha = .01$. (**L4**)

5. **External corpus.** Same as link L1, but the empirical probabilities are measured from an additional set of noisy samples, generated automatically by a statistical parser. Specifically, the data is taken from the BLIPP corpus of Wall Street Journal sentences parsed with Charniak's parser. (**L5**)

6. **V → V.** Transitions from ${}^{h}w_1$ to ${}^{h}w_2$ with probability proportional to their distributional similarity with respect to dependents they take. This is defined more precisely in §4.6.4. (**L6**)

7. **N → N.** Analogously to the previous link type, these are transitions among nouns with probability proportional to their distributional similarity with respect to heads they modify. (**L7**)

8. **V → V.** Transitions from ${}^{h}w_1$ to ${}^{h}w_2$ with probability proportional to their distributional similarity over noun objects when modified by p. (**L8**)

We also used the following final links which add over-smoothed back-off distributions at the end. These represent all levels in a linear back-off sequence estimated

from the training corpus, and a single level of back-off from the additional corpus of noisy samples. Note that these distributions are the same for every state:

9-12. **Backoff1 through Backoff4** Transitions to ${}^{d}w_2$ with probability proportional to $P(N_2 = w_2|P, va), P(N_2 = w_2|va), P(N_2 = w_2|.)$, and uniform respectively. (**L9**,**L10**,**L11**,**L12**)

13. **Backoff5.** Transitions to ${}^{d}w_2$ with probability proportional to $\hat{P}(N_2 = w_2|P, va)$ estimated from the additional noisy corpus. (**L13**)

Additionally, we add identity links (self-loops), to avoid situations where no link type can be followed.

For the Markov Chains estimating the distributions over verbs $p(V|Att, P)$ there is one fewer level of back-off.

The Baseline as a Markov Chain Model

From the description of the link types and the form of our Markov chains, it is clear that our baseline model – BASELINE, described in §4.5.1 – is a special case of a Markov chain model, where the maximum degree of the walks is limited to $d = 1$ and the only link types allowed are the relative frequency link L1 and the back-off links L9 through L12. Thus the models for estimating the distributions over N_1 and N_2 have five link types each, and the models for estimating distributions over V have four link types each (one less, because there is one less level of backoff).

If the maximum degree of the walk is $d = 1$, then there are no parameters for weighting the different possible number of steps. Also, because only one transition is taken, this is a transition according to the final distribution P^{fin}. The estimated distribution for N_2 given va, P, V in this case is $p(N_2|va, P, V) = p_{p,va,fin}(N_2|V)$. The parameters are thus the same as illustrated in Figure 4.4 for the BASELINE model.

4.6.4 Random Walk Experiments

The training, development and test sets we use were described in §4.4.1. Our algorithm uses the training set to estimate empirical distributions and the development set to train the parameters of the random walk. We report accuracy results on the final test set. All results reported here are on the final test set and not the preliminary test set PreTest, described in §4.6.

In addition to this training data set, we generate additional much noisier training data, using the BLLIP corpus. This data is used for defining the link types L5,L6,L7,L8, and L13. BLLIP is a corpus of 1,796,386 automatically parsed English sentences (Charniak, 2000), which is available from the Linguistic Data Consortium (www.ldc.upenn.edu). From the parsed sentences, we extracted tuples of four headwords and attachment site for ambiguous verb or noun PP attachments. We filtered out all tuples that contained one or more words not occurring in the training, test, or development sets. This made for a total of 567,582 tuples. We will call this dataset BLLIP-PP. One can expect this data to be rather noisy, since PP attachment is one of the weakest areas for state of the art statistical parsers. For the parser (Collins, 1999), an accuracy of about 82.29/81.51 recall/precision is reported for any dependency where the modifier is a PP, including unambiguous cases. Note that the parser uses a wider context (mainly structural) from the sentences's parse tree. An automatically parsed corpus has been used before in an unsupervised prepositional phrase attachment system (Pantel and Lin, 2000).

Figure 4.8 presents the results of our experiments. The Baseline and C&B accuracy figures are repeated from Figure 4.6. The Baseline here uses $\sigma = 128$ for the Gaussian prior as before, and a single equivalence class for the interpolation parameters.

Next we describe the incremental addition of links to our model, with discussion of the performance achieved. For all models using Markov Chains, we fix the maximum degree of the walks for estimating the uni-lexical dependencies $P(v|p, Att)$ to $d = 2$, and the maximum degree of all other walks, estimating bi-lexical dependencies, to

Figure 4.8: Summary of results on the final test set of 3,097 samples. In the significance column > means at level .05 and ≫ means at level .005.

		Model	Link Types	Degree	Accuracy	Signif
Baselines	1	C&B			84.18%	
	2	C&B + stem verbs			84.50%	not > 1
	3	C&B on BLLIP-PP			85.53%	> 1
	4	Baseline	L1,L8,L9,L10,L11	1,1	85.89%	≫ 1
	5	Baseline + stem verbs	L1,L8,L9,L10,L11	1,1	86.05%	≫ 2
Random	6	Morph Verbs	+ L2Verbs	2,3	86.08%	not > 4, ≫ 2
Walks	7	Morph Verbs and Nouns	+ L2Nouns	2,3	86.18%	not > 4, ≫ 2
	8	Morph & Syn	+L3Verbs,L3Nouns	2,3	86.53%	not > 4, ≫ 2
	9	Morph & Syn & Back-links	+L4	2,3	86.57%	not > 4, ≫ 2
	10	Morph & Syn & Back-links & Sim_{JS_β}	+L6,L7	2,3	86.89%	≫ 2, >3,> 4
	11	Morph & Syn & Back-links & Sim_{JS_β} & BLIPP-PP	+L5,L13	2,3	87.15%	≫ 3,> 4
	12	Final	see text	2,3	87.54%	> 9,≫ 3, ≫ 4

$d = 3$.[5]

Figure 4.8 has six columns. The first column is used for numbering the models; the second column lists a short description of the model; the third column lists the types of links used in the corresponding model; the fourth column specifies the maximum degrees of the Markov Chains for estimating the uni-lexical distributions (over verbs V), and the bi-lexical ones (over first and second nouns N_1, N_2). The fifth column lists the accuracy on the final test set, and the sixth column lists results of relevant statistical significance tests. The significance tests are McNemar's test using the normal approximation.

1. **Morphology.** The morphology links are L2Verbs for verbs and L2Nouns for nouns. The link between verbs (L2Verbs) was helpful. This link was added to the Markov Chains for estimating $p(V|Att, p)$, $p(N_1|Att, v, p)$, and $p(N_2|va, v, p)$. The accuracy on the test set was 86.08%, as shown in row six of Figure 4.8. To compare the gain from morphology achieved through random walks to the gain achieved if we pre-process the training data to stem the verbs and use the Baseline model, we list the results of the latter experiment in

[5]For computational reasons, we have only explored paths of maximum degree three for models with many features. For smaller models, using higher degrees showed a slight gain. Thoroughly investigating the contribution of longer walks is left to future research.

Line five of the figure (repeated from Figure 4.6). The gain from verb morphology when using random walks is a bit higher than the gain when stemming the training data. Neither of the gains is statistically significant. Adding noun morphology as well was also helpful as can be seen in row seven of the figure.

2. **WordNet Synsets.** As described in the definition of the synonym links (L3Verbs and L3Nouns), we use WordNet in a simple way – for every word, we find its top three most common senses, and make a link from the word to all other words having those senses. The transition probability is proportional to the smoothed number of occurrences of the target word. Thus the link is not symmetric. We obtained accuracy gains from adding the synonym links, as can be seen in row eight of the figure. The accuracy of a random walk model using morphology and synonyms jumped to 86.53%. However, the difference between this model and BASELINE is not significant.

3. **Back Link.** Line nine shows a model which adds one more link type to the model of Line eight – this a back link from dependents to heads, link type L4. The accuracy of the model increased very slightly – from 86.53% to 86.57%. We will show in the discussion section that adding this link amounts to incorporating the cooccurrence smoothing method of (Essen and Steinbiss, 1992).

4. **Similarity based on Jensen-Shannon divergence.** We add links between states with the same memory bit with transition probabilities proportional to their distributional similarity. For the sake of concreteness, consider a random walk for estimating $p(N_2|va, p, v)$. Let q_{v_i} denote the empirical distribution of dependents of the preposition p modifying verb v_i: $\hat{p}(N_2|p, v_i, va)$ estimated from the BLLIP-PP corpus. We define a similarity function between verbs $sim_{JS_\beta}(v_1, v_2) = \exp(-\beta JS(q_{v_1}, q_{v_1}))$. JS stands for *Jensen-Shannon divergence* between two probability distributions (Rao, 1982) and is defined in terms of the KL *divergence* D as:

$$JS(q_1, q_2) = \frac{1}{2}\{D(q_1||\frac{q_1+q_2}{2}) + D(q_2||\frac{q_1+q_2}{2})\}$$

The same similarity function was used in (Dagan et al., 1999; Lee, 1999). We

add a link from verbs to verbs (link type **L6**) that has transitions from each verb, to its top K closest neighbors in terms of the similarity sim_{JS_β}. In our experiments, β was 50, and K was 25. We did not extensively fit these parameters. It would be possible to also fit β by including it as a model parameter and differentiating with respect to it, but we did not implement this solution. The transition probability is the normalized value of the similarity. Similarly we add links between nouns based on their similarity sim_{JS_β} with respect to the empirical distribution $\hat{P}(V|va, p, n)$ in BLLIP-PP (link type **L7**). Crucially these similarities are estimated from the additional noisy data in BLLIP-PP. Estimating such distributional similarities from the training data was not nearly as helpful, possibly due to sparseness.

Up until now we have been discussing the $p(N_2|va, p, v)$ dependency distribution. For the other dependency relations distributions – $p(N_2|na, p, n_1)$, and $p(N_1|Att, p, v)$, we similarly add links between the heads based on their sim_{JS_β} with respect to the empirical distribution of their dependents in BLLIP-PP, and between the dependents proportional to their similarity sim_{JS_β} of head distributions. The accuracy of the resulting model, when these links are added is shown in row ten of Figure 4.8. This model adds only the distributional similarity links to the model in row nine. Its accuracy was 86.89%, which is the first significant improvement over the Baseline model. This model also significantly outperforms the C&B model trained on the union of the original training set and BLIPP-PP.

5. **Empirical distribution links from BLIPP-PP.** The model in line 11 adds empirical distribution links from heads to dependents, based on noisy BLIPP-PP additional training samples. The accuracy of this model is a bit higher than the previous model – 87.14%, but it turns out the model may be overfitting with this many link types. As we will see for the Final Model in line 12, it is possible to do better by removing links from this model.

6. **Final Model.** The final model includes the links from the Baseline, L5, L13, morphology for verbs, and the previously discussed sim_{JS_β} links. In addition,

one more kind of sim_{JS_β} links was added – L8. The final model had an accuracy of 87.54%, which is close to the upper bound. This model is obtained from the previous model in line 11, by removing the verb and noun synonym links, the noun morphology links, and adding the L8 links. The superiority of this model to the previous one is possibly due to overfitting or local maxima. Finding ways around local maxima is a subject of future research.

Other algorithms can also make use of additional noisy training data. We ran the C&B algorithm on the union of the training data and BLIP-PP and its accuracy was also improved as shown in row two of the figure. However, the random walk model is able to make better use of the additional noisy data, as it learns suitable weights for the estimates obtained from it.

Attempts in the past to use additional unsupervised data to improve lexical estimates for statistical parsers have been relatively unsuccessful. For example Charniak (1997) reports an experiment where 30 million words of text were parsed automatically and added as additional training data for a parser estimating lexical dependency probabilities. The results showed a rather small accuracy increase – 0.5% precision and recall. Taken in this context, the success of the random walk model at fruitfully incorporating such data is encouraging.

4.6.5 Extended Example

We describe in more detail the link types in the model in Line 10 of Figure 4.8, and the parameter settings that were learned from data. We also give examples of inference paths that were followed for deriving probability estimates for test set quadruples.

The model in Line 10 includes morphology and synonym links, as well as backlinks and distributional similarity links. We describe the parameters learned for each of the Markov Chains corresponding to different types of dependency relations. The λ_i parameters are grouped into a single equivalence class. The γ_{time} parameters fall into two equivalence classes, depending on the observed count of the pair (*begin*, *end*), as previously described.

Link Type	$Att = va$		$Att = na$	
	Non-Final Weight	Final Weight	Non-Final Weight	Final Weight
$P \overset{L1}{\rightarrow} V$	1.000	0.009	1.000	0.002
$P \overset{L9}{\rightarrow} V$	-	0.175	-	0.007
$P \overset{L10}{\rightarrow} V$	-	0.013	-	0.326
$P \overset{L11}{\rightarrow} V$	-	0.803	-	0.665
$V \overset{L2VerbsMorph}{\rightarrow} V$	-	0.700	-	0.885
$V \overset{L3VerbsSynn}{\rightarrow} V$	-	0.148	-	0.104
$V \overset{Identity}{\rightarrow} V$	-	0.152	-	0.011
EqClass	$time = 0$	$time = 1$	$time = 0$	$time = 1$
$c(end, begin) = 0$	0.922	0.078	0.799	0.201
$c(end, begin) > 0$	0.531	0.469	0.838	0.162

Figure 4.9: Link types and estimated parameters for $p(V|Att, P)$ random walks.

- The Markov Chains for estimating $p(V|Att, P)$ have maximum degree two. Two pairs of transition matrices are learned for this type of walk – one for verbal attachment, and one for noun attachment. Figure 4.9 shows the link types and the estimated values of the parameters for the verb and noun attachment type.

- Generation of N_1. The Markov Chains for estimating $p(N_1|Att, P, V)$ are similar to the Markov Chains for $p(N_2|va, P, V)$, which we have been using as example throughout. The maximum degree of these chains is $d = 3$. Figure 4.10 shows the link types and link weights for the two types of attachments. While each preposition p has its own pair of transition matrices, the link weights are tied across prepositions. We thus have two pairs of link weights, one pair (non-final, final) for each attachment type, as for the $p(V|Att, P)$ chains.

- Generation of N_2. Figure 4.11 shows the parameters of the $p(N_2|va, P, V)$ and $p(N_2|na, P, N_1)$ walks. In this case the link types for the two attachment types are not exactly the same because for verb attachment the head is a verb, and for noun attachment the head is a noun.

A test set case which this random walks model classified correctly and the BASELINE did not is, for example, the tuple:

Link Type	$Att = va$			$Att = na$		
	Non-Final Weight		Final Weight	Non-Final Weight		Final Weight
$V \overset{L1EmpDistr}{\rightarrow} N_1$	0.580		0.266	0.826		0.003
$V \overset{L9Backoff1}{\rightarrow} N_1$	-		0.005	-		0.061
$V \overset{L10Backoff2}{\rightarrow} N_1$	-		0.071	-		0.016
$V \overset{L11Backoff3}{\rightarrow} N_1$	-		0.062	-		4.0E-5
$V \overset{L12Backoff4}{\rightarrow} N_1$	-		0.596	-		0.920
$N_1 \overset{L4BackLink}{\rightarrow} V$	0.865		-	0.909		-
$V \overset{L6JS_\beta}{\rightarrow} V$	0.188		-	0.143		-
$N_1 \overset{L7JS_\beta}{\rightarrow} N_1$	0.002		0.006	0.001		0.081
$V \overset{L2VerbsMorph}{\rightarrow} V$	0.009		-	0.028		-
$N_1 \overset{L2NounsMorph}{\rightarrow} N_1$	0.010		0.036	0.003		0.060
$V \overset{L3VerbsSynn}{\rightarrow} V$	0.217		-	0.000		-
$N_1 \overset{L3NounsSynn}{\rightarrow} N_1$	0.115		0.902	0.087		0.824
$V \overset{Identity}{\rightarrow} V$	0.006		-	0.001		-
$N_1 \overset{Identity}{\rightarrow} N_1$	0.007		0.055	0.000		0.033
EqClass	$time = 0$	$time = 1$	$time = 2$	$time = 0$	$time = 1$	$time = 2$
$c(end, begin) = 0$	0.380	0.025	0.585	0.090	0.043	0.866
$c(end, begin) > 0$	0.001	0.005	0.994	6.2E-4	6.7E-4	0.999

Figure 4.10: Link types and estimated parameters for $p(N_1|Att, P, V)$ random walks.

v:carry n_1:fight p:against n_2:imperialists Att:noun

This case is incorrectly classified by the C&B model as well. The correct attachment is noun attachment, but no triple or quadruple that contains the preposition "against" occurs in training. Only one pair containing the preposition occurs in the training set – (n_1:fight p:against), and it has a verb attachment, in:

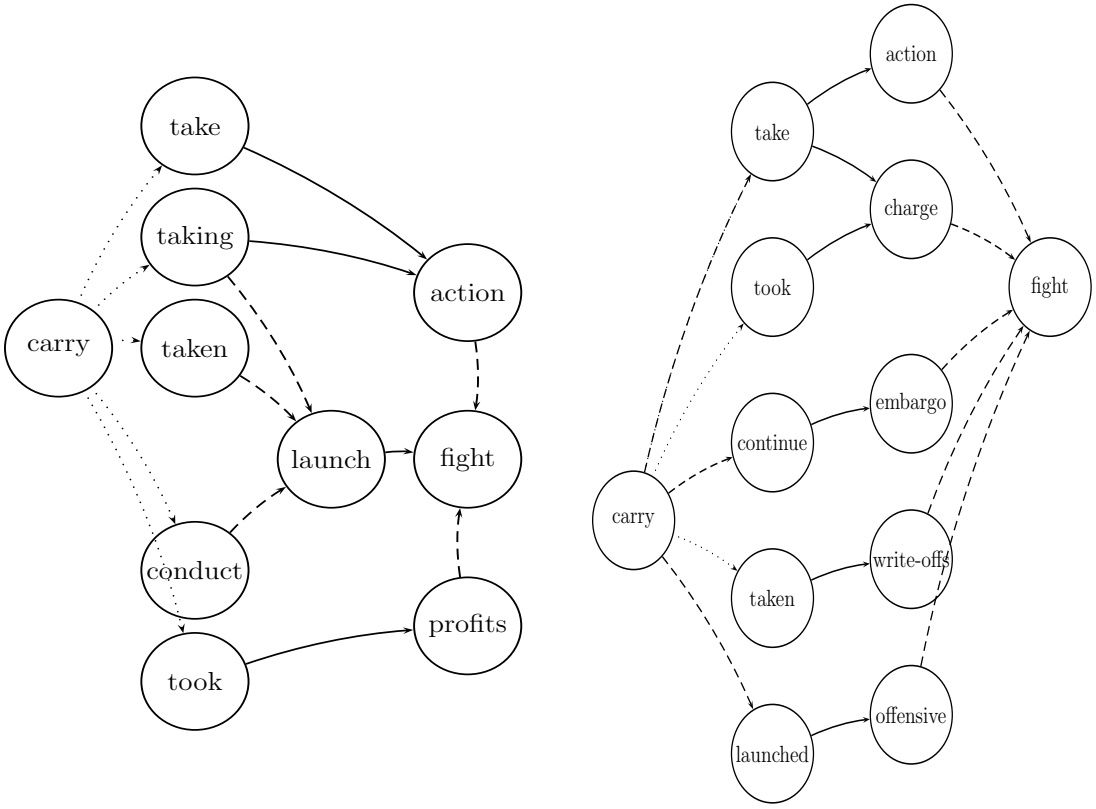
v:launch n_1:fight p:against n_2:board Att:verb

Incidentally, this training set example shows the level of noise or indeterminacy in the training data. This particular tuple could have arguably been assigned noun attachment. Hindle and Rooth (1993) discuss such cases of prepositional pharse attachment indeterminacy. The random walks model classified the above test set example correctly. The Markov Chain for generating the first noun "fight", $P(N_1 = \textit{fight}|Att, P = \textit{agains}, V = \textit{carry})$, found multiple paths from the start state to the end state (not using backoff links). The noun "fight" received non-zero probability under

$Att = va$			$Att = na$		
Link Type	Non-Final Weight	Final Weight	Link Type	Non-Final Weight	Final Weight
$V \overset{L1EmpDistr}{\rightarrow} N_2$	0.691	1.7E-4	$N_1 \overset{L1EmpDistr}{\rightarrow} N_2$	0.504	0.770
$V \overset{L9Backoff1}{\rightarrow} N_2$	-	0.005	$N_1 \overset{L9Backoff1}{\rightarrow} N_2$	-	0.003
$V \overset{L10Backoff2}{\rightarrow} N_2$	-	0.272	$N_1 \overset{L10Backoff2}{\rightarrow} N_2$	-	0.084
$V \overset{L11Backoff3}{\rightarrow} N_2$	-	0.574	$N_1 \overset{L11Backoff3}{\rightarrow} N_2$	-	0.097
$V \overset{L12Backoff4}{\rightarrow} N_2$	-	0.149	$N_1 \overset{L12Backoff4}{\rightarrow} N_2$	-	0.045
$N_2 \overset{L4BackLink}{\rightarrow} V$	0.065	-	$N_2 \overset{L4BackLink}{\rightarrow} N_1$	0.040	-
$V \overset{L6JS_\beta}{\rightarrow} V$	0.071	-	$N_1 \overset{L6JS_\beta}{\rightarrow} N_1$	0.354	-
$N_2 \overset{L7JS_\beta}{\rightarrow} N_2$	0.003	0.769	$N_2 \overset{L7JS_\beta}{\rightarrow} N_2$	0.913	0.053
$V \overset{L2VerbsMorph}{\rightarrow} V$	0.236	-	$N_1 \overset{L2NounsMorph}{\rightarrow} N_1$	0.084	-
$N_2 \overset{L2NounsMorph}{\rightarrow} N_2$	0.001	0.176	$N_2 \overset{L2NounsMorph}{\rightarrow} N_2$	0.015	5.7E-4
$V \overset{L3VerbsSynn}{\rightarrow} V$	1.6E-4	-	$N_1 \overset{L3NounsSynn}{\rightarrow} N_1$	0.053	-
$N_2 \overset{L3NounsSynn}{\rightarrow} N_2$	0.930	0.048	$N_2 \overset{L3NounsSynn}{\rightarrow} N_2$	0.012	0.944
$V \overset{Identity}{\rightarrow} V$	6.3E-4	-	$N_1 \overset{Identity}{\rightarrow} N_1$	0.003	-
$N_2 \overset{Identity}{\rightarrow} N_2$	3.4E-4	0.006	$N_2 \overset{Identity}{\rightarrow} N_2$	0.020	0.002

EqClass	$time = 0$	$time = 1$	$time = 2$	EqClass	$time = 0$	$time = 1$	$time = 2$
$c(end, begin) = 0$	0.002	8.6E-4	0.997	$c(end, begin) = 0$	0.077	0.089	0.833
$c(end, begin) > 0$	0.999	4.2E-4	9.1E-4	$c(end, begin) > 0$	3.4E-4	0.992	0.007

Figure 4.11: Link types and estimated parameters for $p(N_2|va, P, V)$ and $p(N_2|na, P, N_1)$ random walks.



(a) Verb attachment (b) Noun attachment

Figure 4.12: The relevant part of the state space for estimating $P(N_1 = \textit{fight}|Att, P = \textit{against}, V = \textit{carry})$. The solid lines are link type L1 (empirical). The dotted lines are verb synonyms and the dashed lines are Sim_{JS_β}.

both the model for verb attachment and for noun attachment. Figures 4.12(a) and 4.12(b) show the collection of paths that led from the start to the end state (excluding backoff links), under the two models – $P(N_1 = \textit{fight}|va, P = \textit{against}, V = \textit{carry})$, and $P(N_1 = \textit{fight}|na, P = \textit{against}, V = \textit{carry})$.

For generating the second noun – "imperialists", the random walk did not find any path from the start state "carry". Non-zero probability was only accumulated through backoff links.

4.7 Discussion

4.7.1 Relation to the Transformation Model of Jason Eisner

We recently became aware that a very similar model was independently developed by Jason Eisner (Eisner, 2001; Eisner, 2002). Eisner's transformation model also learns a target distribution as the stationary distribution of an appropriately defined Markov chain. The transition probabilities of the Markov chain are also learned from data through maximizing the penalized data log-likelihood. A Gaussian prior on the model parameters was used as well. There are small differences between our model and Eisner's which we outline here.

The Markov chain's transition probabilities do not have the special form of Equation 4.1, repeated here:

$$p(S_t|S_{t-1}) = \gamma p_0(S_t) + (1-\gamma)p'(S_t|S_{t-1})$$

Rather, there is a special halting state HALT, and every state has a separate probability of transitioning to HALT. In this respect, a model with a constant probability γ of transitioning to a halting state is a special case. Eisner notes that the PageRank algorithm (Brin and Page, 1998) is a special case of his framework, where there are no learnable parameters.

Our method of defining halting probabilities is different from Eisner's and it is

not a special case of his method, because as described in Section 4.6, we learn a general multinomial distribution over the number of steps before halting, rather than a geometric one.

The specific application in (Eisner, 2001) was the estimation of distributions over subcategorization frames of verbs for parsing, e.g., $P(NPhungNPPP|S, hung)$, meaning the probability that a sentence symbol S headed by the verb *hung* will expand as an NP to the left of the verb and another NP to the right, followed by a PP. The sentence *I hung the painting with a peg* is an example occurrence of this subcategorization frame for the verb *hung*. The states of the Markov chain over which a probability distribution is estimated are subcategorization frames such as $[NPhungNPPP]$.

There are several other differences compared to our model. The transition distribution is not defined as a linear combination of pre-specified transition distributions obtained from separate knowledge sources. Rather, the link structure is specified using the intuition that subcategorization frames differing only slightly, e.g. by a single insertion or deletion, should be connected by a link. Each link ${l^{ij}}$between two states s_i and s_j is parameterized by a set of features ${f_1}^{ij}, \ldots, {f_k}^{ij}$, with weights ${\theta_1}^{ij}, \ldots, {\theta_k}^{ij}$. The transition probability $P(s_j|s_i)$ is defined through a log-linear model using these features as follows: $P(s_j|s_i) = \frac{e^{\sum_{s=1}^{k} {\theta_s}^{ij}}}{Z(s_i)}$. All parameters θ are learned from the same training set, whereas we use a training set to learn relative frequency transition probability estimates and a development set to fit interpolation parameters.

Several algorithms for computing the estimated distribution and its gradient with respect to the parameters are proposed in (Eisner, 2001). These include algorithms for approximate and exact estimation. If a limited number of time steps d is used for approximation, the derivatives can be computed in time linear in d. Because we learn a general multinomial distribution over number of steps of the walk, our algorithm is quadratic in d. Eisner (2001) mentions exact algorithms for computing the stationary distribution, including matrix inversion in time cubic in the size of the state space (Press et al., 1992) and a faster bi-conjugate gradient method (Press et al., 1992; Greenbaum, 1997).

4.7.2 Relation to Other Models and Conclusions

Random walk models provide a general framework for unifying and combining various notions of similarity-based smoothing. A walk of length one is just a linear interpolation, with interpolation weights typically set empirically as we do here (with the difference that we train to maximize conditional rather than joint likelihood).

A walk of length three following exactly one forward link (like **L1**), followed by one backward link (like **L4**), and another forward link gives exactly the same estimate as cooccurrence smoothing (Essen and Steinbiss, 1992; Lee, 1999). In cooccurrence smoothing, the smoothed probability $\tilde{P}(d|h)$ of a dependent word given a head word is defined as follows:[6]

$$\tilde{P}(d|h) = \sum_{h',d'} \hat{P}(d'|h)\hat{P}(h'|d')\hat{P}(d|h')$$

We can see that this corresponds to summing over all paths of length three in a random walk model where we have a single outgoing link type from every state – a link from heads to dependents, and an inverse link from dependents to heads. Our Markov Chain models contain much richer types of links.

A walk of length two using only one kind of similarity between head states and forward relative frequency links, is similar to distributional similarity smoothing (Dagan et al., 1999). The probability distribution defined using distributional similarity is defined as follows:

$$P_{SIM}(d|h) = \sum_{h' \in S(h)} \hat{P}(d|h') \frac{e^{-\beta D(h||h')}}{\sum_{h'' \in S(h)} e^{-\beta D(h||h'')}}$$

Here $D(\cdot||\cdot)$ is Kullback-Leibler (KL) divergence and $S(h)$ is a set of head words which are closest. In later work, D was substituted with the symmetric Jensen-Shannon divergence (Lee, 1999), as we do here for defining our distributional similarity links L6 and L7. We can see that this expression is summing over walks of length two where we first take a step according to a distributional similarity link between

[6]The original presentation was for bigram language models.

head words and then a relative frequency link from heads to dependents.

The random walks framework that we propose is much more general. A multitude of link types can be defined in it, and they are automatically weighted by the learning algorithm. Paths of shorter and longer lengths can be followed (though the most highly contributing paths are the shorter ones). The generality of this approach to similarity-based smoothing not only gives a high performance prepositional phrase attachment system, but holds the promise of *learning* complex but effective random walk models in other domains. An exciting immediate application of the framework is for estimating more general kinds of word dependency relationships, such as other bi-lexical dependencies required by a lexicalized statistical parser.

Bibliography

[Abney1997] Steven Abney. 1997. Stochastic Attribute-Value Grammars. *Computational Linguistics*, 23(4):597–618.

[Alshawi1992] Hiyan Alshawi, editor. 1992. *The Core Language Engine*. MIT Press.

[Baker et al.1998] Collin Baker, Charles Fillmore, and John Lowe. 1998. The Berkeley Framenet project. In *Proceedings of COLING-ACL-1998*.

[Baldridge and Osborne2003] Jason Baldridge and Miles Osborne. 2003. Active learning for HPSG parse selection. In *Proceedings of the 7th Conference on Natural Language Learning*.

[Baldridge and Osborne2004] Jason Baldridge and Miles Osborne. 2004. Active learning and the total cost of annotation. In *EMNLP*.

[Baum1972] L. E. Baum. 1972. An inequality and associated maximization technique in statistical estimation of probabilistic functions of a markov process. *Inequalities*, 627(3):1–8.

[Berg et al.1984] Christian Berg, Jens Christensen, and Paul Ressel. 1984. *Harmonic analysis on semigroups: theory of positive definite and related functions*. Springer.

[Bikel2004] Daniel Bikel. 2004. A distributional analysis of a lexicalized sttaistical parsing model. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

[Bod1998] Rens Bod. 1998. *Beyond Grammar: An Experience Based Theory of Language*. CSLI Publications.

[Brémaud1999] Pierre Brémaud. 1999. *Markov Chains: Gibbs Fields, Monte Carlo Simulation, and Queues*. Springer-Verlag.

[Bresnan2001] Joan Bresnan. 2001. *Lexical-Functional Syntax*. Blackwell.

[Brill and Resnik1994] E. Brill and P. Resnik. 1994. A rule-based approach to prepositional phrase attachment disambiguation. In *Proceedings of COLING*.

[Brin and Page1998] Sergey Brin and Lawrence Page. 1998. The anatomy of a large-scale hypertextual Web search engine. *WWW7/Computer Networks and ISDN Systems*, 30(1–7):107–117.

[Brown et al.1992] Peter F. Brown, Vincent J. Della Pietra, Peter V. deSouza, Jennifer C. Lai, and Robert L. Mercer. 1992. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4):467–479.

[Carreras and Màrquez2004] Xavier Carreras and Luís Màrquez. 2004. Introduction to the CoNLL-2004 shared task: Semantic role labeling. In *Proceedings of CoNLL*.

[Carreras and Màrquez2005] Xavier Carreras and Luís Màrquez. 2005. Introduction to the CoNLL-2005 shared task: Semantic role labeling. In *Proceedings of CoNLL*.

[Charniak and Johnson2005] Eugene Charniak and Mark Johnson. 2005. Coarse-to-fine n-best parsing and MaxEnt discriminative reranking. In *Proceedings of the 43rd Meeting of the Association for Computational Linguistics*.

[Charniak1997] Eugene Charniak. 1997. Statistical parsing with a context-free grammar and word statistics. In *Proc. 14th National Conference on Artificial Intelligence*, pages 598 – 603.

[Charniak2000] Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *Proceedings of NAACL*, pages 132–139.

[Chen and Goodman1996] Stanley F. Chen and Joshua T. Goodman. 1996. An empirical study of smoothing techniques for language modeling. In *Proceedings of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics*, pages 310–318.

[Chen and Goodman1998] Stanley F. Chen and Joshua T. Goodman. 1998. An empirical study of smoothing techniques for language modeling. Technical Report TR-10-98, Harvard University.

[Chen and Vijay-Shanker2000] John Chen and K. Vijay-Shanker. 2000. Automated extraction of TAGs from the Penn Treebank. In *Proceedings of the 6th International Workshop on Parsing Technologies*.

[Chiang2000] David Chiang. 2000. Statistical parsing with an automatically-extracted tree adjoining grammar. In *Proceedings of the 38th Meeting of the Association for Computational Linguistics*.

[Clark and Curran2004] Stephen Clark and James R. Curran. 2004. Parsing the WSJ using CCG and Log-linear models. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics*.

[Cohn and Blunsom2005] Trevor Cohn and Philip Blunsom. 2005. Semantic role labelling with tree conditional random fields. In *Proceedings of CoNLL shared task*.

[Collins and Brooks1995] Michael Collins and James Brooks. 1995. Prepositional attachment through a backed-off model. In *Proceedings of the Third Workshop on Very Large Corpora*, pages 27–38.

[Collins and Duffy2001] Michael Collins and Nigel Duffy. 2001. Convolution kernels for natural language. In *Proceedings of NIPS*.

[Collins and Duffy2002] Michael Collins and Nigel Duffy. 2002. New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted

perceptron. In *Proceedings of the 40th Meeting of the Association for Computational Linguistics*.

[Collins1997] Michael Collins. 1997. Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Meeting of the Association for Computational Linguistics and the 7th Conference of the European Chapter of the ACL*, pages 16 – 23.

[Collins1999] Michael Collins. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania.

[Collins2000] Michael Collins. 2000. Discriminative reranking for natural language parsing. In *Proceedings of the 17th International Conference on Machine Learning (ICML)*, pages 175–182.

[Collins2001] Michael Collins. 2001. Parameter estimation for statistical parsing models: Theory and practice of distribution-free methods. In *IWPT*. Paper written to accompany invited talk at IWPT 2001.

[Copestake et al.1999] Ann Copestake, Daniel P. Flickinger, Ivan A. Sag, and Carl Pollard. 1999. Minimal Recursion Semantics. An introduction. Ms., Stanford University.

[Crammer and Singer2001] Koby Crammer and Yoram Singer. 2001. On the algorithmic implementation of kernel-based vector machines. *Journal of Machine Learning Research*, 2(5):265–292.

[Crammer and Singer2003] Koby Crammer and Yoram Singer. 2003. Ultraconservative online algoritms for multiclass problems. *Journal of Machine Learning Research*.

[Dagan et al.1999] Ido Dagan, Lillian Lee, and Fernando Pereira. 1999. Similarity-based models of cooccurrence probabilities. *Machine Learning*, 34(1–3):43–69.

[Dietterich1998] Thomas G. Dietterich. 1998. Approximate statistical test for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1923.

[Dowty1991] David R. Dowty. 1991. Thematic proto-roles and argument selection. *Language*, 67(3).

[Eisner2001] Jason Eisner. 2001. *Smoothing a Probabilistic Lexicon via Syntactic Transformations*. Ph.D. thesis, University of Pennsylvania.

[Eisner2002] Jason Eisner. 2002. Transformational priors over grammars. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 63–70.

[Essen and Steinbiss1992] Ute Essen and Volker Steinbiss. 1992. Cooccurrence smoothing for stochastic language modeling. In *ICASSP*, volume 1, pages 161–164.

[Everitt1977] B. S. Everitt. 1977. *The analysis of contingency tables*. Chapman and Hall.

[Fillmore1968] Charles J. Fillmore. 1968. The case for case. In E. W. Bach and R. T. Harms, editors, *Universals in Linguistic Theory*, pages 1–88. Holt, Rinehart & Winston.

[Fillmore1976] Charles J. Fillmore. 1976. Frame semantics and the nature of language. *Annals of the New York Academy of Sciences: Conference on the Origin and Development of Language and Speech*, 280:20–32.

[Gärtner et al.2002] Thomas Gärtner, John W. Lloyd, and Peter A. Flach. 2002. Kernels for structured data. In *Inductive Logic Programming*, pages 66–83.

[Gildea and Hockenmaier2003] Daniel Gildea and Julia Hockenmaier. 2003. Identifying semantic roles using Combinatory Categorial Grammar. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

[Gildea and Jurafsky2002] Daniel Gildea and Daniel Jurafsky. 2002. Automatic labeling of semantic roles. *Computational Linguistics*, 28(3):245–288.

[Gildea and Palmer2002] Daniel Gildea and Martha Palmer. 2002. The necessity of parsing for predicate argument recognition. In *Proceedings of the 40th Meeting of the Association for Computational Linguistics*.

[Good1953] I. J. Good. 1953. The population frequencies of species and the estimation of population parameters. *Biometrika*, 40(16):237–264.

[Goodman2001] Joshua T. Goodman. 2001. A bit of progress in language modeling. In *MSR Technical Report MSR-TR-2001-72*.

[Greenbaum1997] Anne Greenbaum. 1997. *Iterative methods for solving linear systems*. Society for Industrial and Applied Mathematics.

[Haghighi et al.2005] Aria Haghighi, Kristina Toutanova, and Christopher D. Manning. 2005. A joint model for semantic role labeling. In *Proceedings of CoNLL-2005 shared task*.

[Harabagiu and Pasca1999] Sanda Harabagiu and Marius Pasca. 1999. Integrating symbolic and statistical methods for prepositional phrase attachment. In *Proceedings of FLAIRS-99*, pages 303–307.

[Haussler1999] David Haussler. 1999. Convolution kernels on discrete structures. In *UC Santa Cruz Technical Report UCS-CRL-99-10*.

[Hindle and Rooth1993] Donald Hindle and Mats Rooth. 1993. Structural ambiguity and lexical relations. *Computational Linguistics*, 19(1):103–120.

[Hockenmaier and Steedman2002a] Julia Hockenmaier and Mark Steedman. 2002a. Acquiring compact lexicalized grammars from a cleaner treebank. In *Proceedings of Third International Conference on Language Resources and Evaluation*.

[Hockenmaier and Steedman2002b] Julia Hockenmaier and Mark Steedman. 2002b. Generative models for statistical parsing with combinatory categorial grammar. In *Proceedings of the 40th Meeting of the Association for Computational Linguistics*.

[Hockenmaier2003a] Julia Hockenmaier. 2003a. *Data and Models for Statistical Parsing with Combinatory Categorial Grammar*. Ph.D. thesis, School of Informatics, University of Edinburgh.

[Hockenmaier2003b] Julia Hockenmaier. 2003b. Parsing with generative models of predicate-argument structure. In *Proceedings of the 41st Meeting of the Association for Computational Linguistics*.

[Hull1996] David Hull. 1996. Stemming algorithms – A case study for detailed evaluation. *Journal of the American Society for Information Science*, 47(1):70–84.

[Jackendoff1972] Ray Jackendoff. 1972. *Semantic Interpretation in Generative Grammar*. MIT Press.

[Jakub Zavrel and Veenstra1997] Walter Daelemans Jakub Zavrel and Jorn Veenstra. 1997. Resolving PP attachment ambiguities with memory-based learning. In *CoNLL*.

[Jelinek and Mercer1980] Frederick Jelinek and Robert L. Mercer. 1980. Interpolated estimation of markov source parameters from sparse data. In *Proceedings of the Workshop on Pattern Recognition in Practice*.

[Joachims1999] Thorsten Joachims. 1999. Making large-scale SVM learning practical. In B. Scholkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*.

[Joachims2002] Thorsten Joachims. 2002. Optimizing search engines using clickthrough data. In *Proceedings of the ACM Conference on Knowledge Discovery and Data Mining (KDD)*.

[Johnson et al.1999] Mark Johnson, Stuart Geman, Stephen Canon, Zhiyi Chi, and Stefan Riezler. 1999. Estimators for stochastic unification-based grammars. In *Proceedings of the 37th Meeting of the Association for Computational Linguistics*.

[Johnson1998] Mark Johnson. 1998. PCFG models of linguistic tree representations. *Computational Linguistics*, 24(4).

[Katz1987] Slava M. Katz. 1987. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *Proceedings of IEEE Transactions on Acoustics, Speech and Signal Processing*, 35(3):400 – 401.

[Klein and Manning2002] Dan Klein and Christopher Manning. 2002. Conditional structure versus conditional estimation in NLP models. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

[Klein and Manning2003] Dan Klein and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of the 41st Meeting of the Association for Computational Linguistics*.

[Kleinberg1998] Jon Kleinberg. 1998. Authoritative sources in a hyperlinked environment. In *9th ACM-SIAM Symposium on Discrete Algorithms*.

[Kneser and Ney1995] Reinhard Kneser and Hermann Ney. 1995. Improved backing-off for m-gram language modeling. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 181–184.

[Kudo and Matsumoto2001] Taku Kudo and Yuji Matsumoto. 2001. Chunking with support vector machines. In *Proceedings of NAACL*.

[Lafferty and Zhai2001] John Lafferty and Chengxiang Zhai. 2001. Document language models, query models, and risk minimization for information retrieval. In *SIGIR*, pages 111–119.

[Lafferty et al.2001] John Lafferty, Andrew McCallum, and Fernando Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of ICML-2001*.

[Lee1999] Lillian Lee. 1999. Measures of distributional similarity. In *37th Annual Meeting of the Association for Computational Linguistics*, pages 25–32.

[Leslie and Kuang2003] Christina Leslie and Rui Kuang. 2003. Fast kernels for inexact string matching. In *COLT 2003*, pages 114–128.

[Levy and Manning2004] Roger Levy and Chris Manning. 2004. Deep dependencies from context-free statistical parsers: correcting the surface dependency approximation. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics*.

[Lidstone1920] G. J. Lidstone. 1920. Note on the general case of the Bayes-Laplace formula for inductive or a posteriori probabilities. *Transactions of the Faculty of Actuaries*, 8:182–192.

[Lodhi et al.2000] Huma Lodhi, John Shawe-Taylor, Nello Cristianini, and Christopher J. C. H. Watkins. 2000. Text classification using string kernels. In *Proceedings of NIPS*, pages 563–569.

[Magerman1995] David M. Magerman. 1995. Statistical Decision-tree models for parsing. In *Proceedings of the 33rd Meeting of the Association for Computational Linguistics*.

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

[Maxwell and Kaplan1993] John T. Maxwell and Ronald M. Kaplan. 1993. The interface between phrasal and functional constraints. *Computational Linguistics*, 19(4):571–590.

[McDonald et al.2005] Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005. Online large-margin training of dependency parsers. In *Proceedings of the 43rd Meeting of the Association for Computational Linguistics*.

[Merlo et al.1997] P. Merlo, M. Crocker, and C. Berthouzoz. 1997. Attaching multiple prepositional phrases: Generalized backed-off estimation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 149–155.

[Miller et al.1996] Scott Miller, David Stallard, Robert Bobrow, and Richard Schwartz. 1996. A fully statistical approach to natural language interfaces. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*.

[Miller1990] George Miller. 1990. WordNet: an on-line lexical database. *International Journal of Lexicography*, 4(3).

[Minnen et al.2001] Guido Minnen, John Carroll, and Darren Pearce. 2001. Applied morphological processing of English. *Natural Language Engineering*, 7(3):207–223.

[Miyao and Tsujii2005] Yusuke Miyao and Jun'ichi Tsujii. 2005. Probabilistic disambiguation models for wide-coverage HPSG parsing. In *Proceedings of the 43rd Meeting of the Association for Computational Linguistics*.

[Miyao et al.2004] Yusuke Miyao, Takashi Ninomiya, and Jun'ichi Tsujii. 2004. Corpus-oriented grammar development for acquiring a Head-driven Phrase Structure Grammar from the Penn Treebank. In *Proceedings of IJCNLP-04*.

[Ng and Jordan2002] Andrew Ng and Michael Jordan. 2002. On discriminative vs. generative classifiers: A comparison of logistic regression and Naive Bayes. In *NIPS 14*.

[Ng et al.2001] Andrew Y. Ng, Alice X. Zheng, and Michael Jordan. 2001. Link analysis, eigenvectors, and stability. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01)*.

[Oepen et al.2002] Stephan Oepen, Kristina Toutanova, Stuart Shieber, Chris Manning, and Dan Flickinger. 2002. The LinGo Redwoods treebank: Motivation and preliminary applications. In *Proceedings of COLING 19*, pages 1253–1257.

[Olteanu and Moldovan2005] Marian Olteanu and Dan Moldovan. 2005. PP-attachment disambiguation using large context. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

[Osborne and Baldbridge2004] Miles Osborne and Jason Baldbridge. 2004. Ensemble-based active learning for parse selection. In *Proceedings of HLT-NAACL*.

[Palmer et al.2005] Martha Palmer, Dan Gildea, and Paul Kingsbury. 2005. The proposition bank: An annotated corpus of semantic roles. *Computational Linguistics*.

[Pantel and Lin2000] P. Pantel and D. Lin. 2000. An unsupervised approach to prepositional phrase attachment using contextually similar words. In *Proceedings of the 38th Meeting of the Association for Computational Linguistics*, pages 101–108.

[Pollard and Sag1994] Carl Pollard and Ivan A. Sag. 1994. *Head-Driven Phrase Structure Grammar*. University of Chicago Press.

[Pradhan et al.2004] Sameer Pradhan, Wayne Ward, Kadri Hacioglu, James Martin, and Dan Jurafsky. 2004. Shallow semantic parsing using support vector machines. In *Proceedings of HLT/NAACL-2004*.

[Pradhan et al.2005a] Sameer Pradhan, Kadri Hacioglu, Valerie Krugler, Wayne Ward, James Martin, and Dan Jurafsky. 2005a. Support vector learning for semantic argument classification. *Machine Learning Journal*.

[Pradhan et al.2005b] Sameer Pradhan, Wayne Ward, Kadri Hacioglu, James Martin, and Daniel Jurafsky. 2005b. Semantic role labeling using different syntactic views. In *Proceedings of the 43rd Meeting of the Association for Computational Linguistics*.

[Press et al.1992] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. 1992. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press.

[Price1990] P. J. Price. 1990. Evaluation of spoken language systems: The ATIS domain. In *Proceedings of the ARPA Human LanguageTechnology Workshop*.

[Punyakanok et al.2004] Vasin Punyakanok, Dan Roth, Wen tau Yih, Dav Zimak, and Yuancheng Tu. 2004. Semantic role labeling via generalized inference over classifiers. In *Proceedings of CoNLL-2004*.

[Punyakanok et al.2005] Vasin Punyakanok, Dan Roth, and Wen tau Yih. 2005. The necessity of syntactic parsing for semantic role labeling. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

[Rao1982] Radhakrishna C. Rao. 1982. Diversity: Its measurement, decomposition, apportionment and analysis. *The Indian Journal of Statistics*, 44(A):1–22.

[Ratnaparkhi et al.1994] Adwait Ratnaparkhi, Jeff Reynar, and Salim Roukos. 1994. A maximum entropy model for prepositional phrase attachment. In *Workshop on Human Language Technology*.

[Riezler et al.2000] Stefan Riezler, Detlef Prescher, Jonas Kuhn, and Mark Johnson. 2000. Lexicalized stochastic modeling of constraint-based grammars using log-linear measures and EM training. In *Proceedings of the 38th Meeting of the Association for Computational Linguistics*, pages 480—487.

[Riezler et al.2002] Stefan Riezler, Tracy H. King, Ronald M. Kaplan, Richard Crouch, John T. Maxwell, III, and Mark Johnson. 2002. Parsing the Wall Street Journal using a Lexical-Functional Grammar and discriminative estimation techniques. In *Proceedings of the 40th Meeting of the Association for Computational Linguistics*.

[Riloff1993] Ellen Riloff. 1993. Automatically constructing a dictionary for information extraction tasks. In *National Conference on Artificial Intelligence*.

[Schank1972] Roger C. Schank. 1972. Conceptual dependency: A theory of natural language understanding. *Cognitive Psychology*, 3:552–631.

[Sha and Pereira2003] Fei Sha and Fernando Pereira. 2003. Shallow parsing with conditional random fields. In *HLT-NAACL*.

[Shen and Joshi2003] Libin Shen and Aravind K. Joshi. 2003. An SVM-based voting algorithm with application to parse reranking. In *Proceedings of CoNLL*, pages 9–16.

[Srinivas and Joshi1999] Bangalore Srinivas and Aravind K. Joshi. 1999. Supertagging: An approach to almost parsing. *Computational Linguistics*, 25:237–265.

[Steedman1996] Mark Steedman. 1996. *Surface Structure and Interpretation.* MIT Press.

[Stetina and Nagao1997] Jiri Stetina and Makoto Nagao. 1997. Corpus based PP attachment ambiguity resolution with a semantic dictionary. In *Proc. 5th Workshop on Very Large Corpora*, pages 66–80.

[Steven Abney1999] Yoram Singer Steven Abney, Robert E. Shapire. 1999. Boosting applied to tagging and PP attachment. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP).*

[Surdeanu et al.2003] Mihai Surdeanu, Sanda Harabagiu, John Williams, and Paul Aarseth. 2003. Using predicate-argument structures for information extraction. In *Proceedings of the 41st Meeting of the Association for Computational Linguistics.*

[Suzuki et al.2003] Jun Suzuki, Tsutomu Hirao, Yutaka Sasaki, and Eisaku Maeda. 2003. Hierarchical directed acyclic graph kernel: Methods for structured natural language data. In *Proceedings of the 41st Meeting of the Association for Computational Linguistics*, pages 32 – 39.

[Taskar et al.2004] Ben Taskar, Dan Klein, Michael Collins, Daphne Koller, and Christopher D. Manning. 2004. Max-margin parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP).*

[Taylor and Karlin1998] H. M. Taylor and S. Karlin. 1998. *An Introduction to Stochastic Modeling.* Academic Press, San Diego, third edition.

[Thompson et al.2003] Cynthia A. Thompson, Roger Levy, and Christopher D. Manning. 2003. A generative model for semantic role labeling. In *Proceedings of ECML-2003*.

[Toutanova and Manning2002] Kristina Toutanova and Christopher D. Manning. 2002. Feature selection for a rich HPSG grammar using decision trees. In *Proceedings of CoNLL*.

[Toutanova et al.2002] Kristina Toutanova, Christopher D. Manning, Stuart Shieber, Dan Flickinger, and Stephan Oepen. 2002. Parse disambiguation for a rich HPSG grammar. In *Proceedings of Treebanks and Linguistic Theories*, pages 253–263.

[Toutanova et al.2003] Kristina Toutanova, Mark Mitchell, and Christopher D. Manning. 2003. Optimizing local probability models for statistical parsing. In *Proceedings of the 14th European Conference on Machine Learning (ECML)*.

[Toutanova et al.2004a] Kristina Toutanova, Christopher D. Manning, and Andrew Y. Ng. 2004a. Random walks for estimating word dependency distributions. In *Proceedings of the 21st International Conference on Machine Learning (ICML)*.

[Toutanova et al.2004b] Kristina Toutanova, Penka Markova, and Chrostopher D. Manning. 2004b. The leaf projection path view of parse trees: Exploring string kernels for HPSG parse selection. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

[Toutanova et al.2005a] Kristina Toutanova, Aria Haghighi, and Christopher D. Manning. 2005a. Joint learning improves semantic role labeling. In *Proceedings of the 43rd Meeting of the Association for Computational Linguistics*.

[Toutanova et al.2005b] Kristina Toutanova, Christopher D. Manning, Dan Flickinger, and Stephan Oepen. 2005b. Stochastic HPSG parse disambiguation using the Redwoods corpus. *Journal of Logic and Computation*.

[Tsochantaridis et al.2004] Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. 2004. Support vector machine learning for interdependent and structured output spaces. In *Proceedings of the 21st International Conference on Machine Learning (ICML)*.

[Vanschoenwinkel and Manderick2003] Bram Vanschoenwinkel and Bernard Manderick. 2003. A weighted polynomial information gain kernel for resolving prepositional phrase attachment ambiguities with support vector machines. In *IJCAI*.

[Vapnik1998] Vladimir Vapnik. 1998. *Statistical Learning Theory*. Wiley, New York.

[Wahlster2000] Wolfgang Wahlster, editor. 2000. *Verbmobil: Foundations of Speech-to-Speech Translation*. Springer.

[Weston and Watkins1998] Jason Weston and Chris Watkins. 1998. Multi-class Support Vector Machines. Technical Report TR-98-04, Department of Computer Science, Royal Holloway, Univeristy of London.

[Witten and Bell1991] Ian H. Witten and Timothy C. Bell. 1991. The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, 37,4:1085–1094.

[Xia1999] Fei Xia. 1999. Extracting tree adjoining grammars from bracketed corpora. In *Proceedings of the 5th Natural Language Processing Pacific Rim Symposium (NLPRS-99)*.

[Xue and Palmer2004] Nianwen Xue and Martha Palmer. 2004. Calibrating features for semantic role labeling. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

[Yi and Palmer2005] Szu-ting Yi and Martha Palmer. 2005. The integration of syntactic parsing and semantic role labeling. In *CoNLL Shared task*.