Exploiting Treebanking Decisions for Parse Disambiguation

by
Md. Faisal Mahbub Chowdhury

M.Sc. Thesis
European Masters Program in *Language and Communication Technologies (LCT)*

Universität des Saarlandes
Free University of Bozen-Bolzano

Supervisors:

Dr. Yi Zhang
PD Dr. Valia Kordoni
Prof. Dr. Hans Uszkoreit

September 2009

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

Saarbrücken, September 2009

Md. Faisal Mahbub Chowdhury

Abstract

Treebanks play an increasing role in computational linguistics for training parsers. Some treebanking environments allow annotators to quickly navigate through the parse forest and identify the correct or incorrect or preferred analysis in the current context by selecting or rejecting discriminants. Although, these treebanking decisions are recorded in log files or databases, but, to our best knowledge, until now nobody has inspected potentiality of incorporating such fine-grained decisions made by human annotators for automatic parse disambiguation. This thesis examines this new potential research direction by developing a novel approach for extracting discriminative features using treebanking decisions. The thesis presents comparative analyses of the performance of discriminative disambiguation models built using the treebanking decision features and the state-of-the-art features which indicate features extracted using treebanking decisions are more efficient and informative compared to their traditional counterparts.

We highlight how these different types of features scale when their corresponding models are tested on out-of-domain data. The result suggests that, treebanking decision features are more robust. Analyses from different perspectives such as impact of different types of decisions on the disambiguation model, or using the disambiguation model of the treebanking decisions feature as a re-ranker are also included.

The study also develops a method to extract patterns of correlated discriminant from human decisions and use them for parse forest reduction. The empirical results indicate that, finding such patterns that yields substantial reduction of parse forest preserving the preferred analyses is not an easy task.

The thesis argues that, the discriminative nature of the treebanking decisions allows them to be highly effective features to contribute to an efficient disambiguation model. This is demonstrated by a number of experiments that also reveal some open research questions for future works.

Acknowledgements

Two years ago, when I landed in Europe, I was confused whether I made the right decision to come here for higher study. Looking back to the memories, many things that I have learnt and the wonderful people that I have met, I am very happy that I took that decision.

I would like to thank Dr. Yi Zhang for being such a wonderful supervisor. I am indebted to him for patiently listening to me in numerous occasions and giving his thoughtful opinions, and especially, for helping me when I was struggling at the beginning with various setup related problems of the Logon system. Thanks to his vast knowledge on this particular field of research, he introduced me to different types of works which might not be directly related to my thesis but surely enriched my knowledge.

I would also like to thank my other supervisor Dr. Valia Kordoni for her encouraging words and support throughout this thesis. Before starting the thesis I almost knew nothing about parsing, and yet she agreed to supervise me. I also thank for her nice emails when my father was seriously ill and I was very upset. I am also grateful to her and Dr. Zhang for contributing in one of my papers. Above all, I would like to thank her for being a very helpful coordinator in these two years.

I am thankful to Prof. Bernardo Magnini for asking me to come to FBK-Irst for internship. I am glad that I took that opportunity. I enjoyed my time there. Many thanks to Matteo Negri for his guidance in that period and also for letting me know various important things regarding technical writing when we did a publication together.

Thanks to Dr. Raffaella Bernardi for her caring and guidance during my stay in Bolzano.

It could have been really tough for me to finish everything in time and properly if there were no Ms. Jutta Springer, Ms. Bobbye Pernice and Ms. Federica Cumer. Whether it was visa problem, course related issue, housing issue or any other problems, they were always there to help me get out of those situations. Thanks to them for their incredible support.

Special thanks goes to my mom, dad and sisters, for supporting me during all these years, for helping me and for always being there for me. I am blessed to have such a wonderful family.

Thanks also to all of my friends (near and far, old and new) for being so helpful and supportive.

Finally, thanks to VoipWise, Gmail and Yahoo! for their wonderful technologies which allowed me always to be in touch with my family and friends, and also to Bolzano and Saarbrücken for being such friendly places to live.

Thank you all very much.

— Faisal

Saarbrücken, September 2009

Contents

Abstract **iv**

Acknowledgements **v**

List of Tables **x**

List of Figures **xi**

1 Introduction **1**

1.1 Background 1

1.2 Motivation and research questions 3

1.3 Contribution of this thesis 4

1.4 Outline of the thesis 5

2 Mathematical Preliminaries **7**

2.1 Entropy 7

2.2 Maximum entropy models 8

2.3 Maximum likelihood estimation 10

3 Previous Related Works **12**

3.1 Statistical parse selection 12

3.2 Discriminant-based treebanking environments 16

3.3 Domain adaptation, re-ranking and self-training 18

4 Treebanking Decisions and Features **20**
4.1 The Redwoods-style treebanks 21
4.2 The state-of-the-art feature types 22
4.3 Treebanking decisions 23
4.4 Why treebanking decisions 24
4.5 Feature extraction using treebanking decisions 25
4.6 How are *TDF*s different from the traditional features? 25

5 System Overview **28**
5.1 Feature extractor 29
5.2 Component for training maximum entropy models and testing 30
5.3 Performance analyser 33
5.4 Re-ranker 33
5.5 Component for parse forest reduction using ranked *TDF*s 34
5.6 *LOGON* System 36

6 Experimentation Environment **37**
6.1 Data 37
6.2 Decisions taken by the human annotators and system 38
6.3 Feature sets 39
6.4 Discriminative models used for the experiments 40
6.5 Evaluation measures 40

7 Results and Analyses **42**
7.1 Comparison of performances among the models of different feature types 43
7.2 Performance measurement on out-of-domain data 43
7.3 Active features 44
7.4 Effect of human annotated decisions, contexts and different decision combinations 45
7.5 *TDF* model as re-ranker 46
7.6 Parse forest reduction using individual top ranked features 47

8 Extracting Correlated Discriminants from Human Decisions 48
8.1 Pattern extraction 49
8.2 Reducing size of the parse forests using patterns 50
8.3 Experiments and observations 51

9 Conclusion 55
9.1 Summary 55
9.2 Comparison to the related works 57
9.3 Open questions and future works 58

Appendix A 61

List of Tables

4.1 Example of the state-of-the-art features extracted from the derivation tree in Figure 4.1. 23

4.2 Example of the *TDF*s extracted from the derivation tree in Figure 4.1 using the treebanking decision *"D4* ***hspec*** *the || dog"* in Figure 4.2 25

7.1 Accuracies obtained on in-domain data using n-grams (n=4), local configurations (with grandparenting level 3), active edges and TDFC. 42

7.2 Accuracies obtained on out-of-domain data. 43

7.3 FHC and FTHC calculated for in-domain data 44

7.4 Accuracies obtained using different combination of *TDF*. *TDF1* is the *TDF*s extracted using template *T1*; *TDFC* is extracted using the combination of templates *T1*, *T2* and *T3*; *YTDFC* is same as *TDFC* except that the features are collected using only annotated and inferred "Yes" decisions; and finally, *ATDFC* is the collection of *TDF*s extracted using only human annotated decisions. 45

7.5 Accuracies obtained by using the disambiguation model of TDFC as re-ranker. 46

7.6 Result of parse forest reduction using *Z%* of top ranked *TDF*s individually. In the table, by "*% of the size of forests reduced*", we mean the percentage of the original number of total parse trees (of all the reduced parse forests) that remain after reduction. 47

8.1 Parse forest reduction using *CGP* and *FGP*. 52

8.2 Impact of patterns with negative and lexical discriminants on parse forest reduction. 53

List of Figures

3.1 The SRI Cambridge Treebanker GUI. 16
3.2 `[incr tsdb()]`treebank annotation GUI. 17

4.1 Example of HPSG derivation tree. Phrasal nodes are labeled with identifiers of grammar rules, and (pre-terminal) lexical nodes with class names for types of lexical entries. Although, the original derivation tree format has pre-terminals corresponding to lexical identifiers, this figure shows a modified format where these identifiers are mapped to one of the abstract lexical types of ERG. 22
4.2 Example forest and discriminants 24

5.0 Steps of feature extraction: *(a)* sample parse trees and treebanking decision, *(b)* corresponding parse trees for the decision (i.e. belonging to the same sentence) is selected, *(c)* presence of the elements of the treebanking decision is identified in one of tree, *(d)* features are extracted using templates. 32
5.1 Work flow of the whole process of parse disambiguation with treebanking decisions. 34
5.2 Process of re-ranking. 35

Chapter 1

Introduction

> *"....specifying the features of a stochastic unification-based grammar (SUBG) is as much an empirical matter as specifying the grammar itself."* (Johnson et al., 1999)

1.1 Background

Parsing, or, more formally, syntactic analysis, is the process of analyzing a sequence of tokens (e.g. sentence) to determine their grammatical structure with respect to a given formal grammar. Parsing is usually considered as an intermediate stage and used to uncover structures that are used by later stages of processing for many natural language processing (NLP) tasks. Early parsing initiatives based on probabilistic models on CFG formalism demonstrate efficient processing but they could not surpass a certain limit of accuracy. This is due to lack of deep linguistic knowledge for CFG formalism (Abney, 1997). Alternative strategies based on deep grammar formalisms such as Head-driven Phrase Structure Grammar (HPSG), Lexical Functional Grammar (LFG), etc., face three main challenges — coverage, efficiency and high ambiguity.

Until only a few years ago, time and memory requirements of unification-based processing systems used to be prohibitive for many applications. Several hundred megabytes of main memory were considered the absolute minimum for parsing medium-complexity input with a large-scale grammar. Parsing times of one or two minutes per sentence were not considered unusual. Over the past few years, however, significant progress in efficient processing has been achieved. (Callmeier, 2001)

For some languages, such as English, German, etc., more and more corpus and

treebanks[1] are becoming available.[2] Large grammars are being built for years for some languages (such as ERG (Flickinger, 2000), GG (Müller and Kasper, 2000), etc.). So, coverage is also increasing.

However, ambiguity (in this thesis we are interested on syntactic ambiguity) remains as one of the critical problem of NLP. Depending on the richness of the grammar, a sentence can have multiple number (sometimes thousands) of syntactic analyses. *Parse disambiguation*, or, *parse selection* is the task of selecting a preferred analysis given an input sentence. There are two possible approaches for parse disambiguation. One approach is rule based where, the strategy is to encode knowledge by define rules, hand-engineered grammars and patterns. But it requires much effort and is not robust. The other widely adopted approach is to treat the problem of disambiguation as a classification task, learn classifiers from labeled training data and build training models to be used on unseen data. While such models can vary based on their underlying mathematical framework and the type of features they use, all of them are basically statistical models. The latter approach is also dependent on underlying grammars, but they are not restricted to the grammars.

While treebanks are mainly used in corpus linguistics for studying syntactic phenomena, they play an increasing role in computational linguistics for training parsers. Previous researches have adopted two approaches to use treebanks for disambiguation models. One approach, known as generative, uses only the gold or preferred parse trees (Ersan and Charniak, 1995; Charniak, 2000). The other approach is to use discriminative models (will be discussed in detail in Chapters 2 and 3), which is created by training maximum entropy model or conditional log-linear model on treebanks (Johnson et al., 1999; Toutanova et al., 2005). The discriminative parse selection models are trained by maximizing the probability of all the preferred analyses relative to all the alternative and non-preferred analyses. This gives us a statistical model for the distribution of parses conditioned on a given input string. In this latter approach, features such as local derivational configurations (i.e. local sub-trees), grandparents, n-grams, etc., are extracted from all the trees and are utilized to build the model. Neither of the approaches considers cognitive aspects of treebanking, i.e. the fine-grained decision-making process of the human annotators.

Although, due to considerable amount of efforts in last 10 years, disambiguation accuracies of the discriminative models have reached a certain point, yet a lot more to do. Some of the treebanks are annotated semi-automatically where explicit deci-

[1] A treebank or parsed corpus is a text corpus in which each sentence has been parsed, i.e. annotated with syntactic structure. Syntactic structure is commonly represented as a tree structure, hence the name *Treebank* is used to refer to such corpus.

[2] See http://nlp.stanford.edu/links/statnlp.html#Corpora for a list of corpora including treebanks.

sions of annotation (we will refer to them as (annotated) treebanking decisions and formally define the notion of *treebanking decision* in Chapter 4) are made by the human annotators through graphical annotation tool are recorded in database or log files (Oepen, 2001). This thesis analyzes the prospect of improvement in automatic parse disambiguation if such decisions are exploited. To our best knowledge, this is the first work which reports on exploiting treebanking decisions for parse disambiguation.

1.2 Motivation and research questions

As mentioned before, in this thesis, we try to address the syntactic ambiguity problem of natural language analysis by utilizing treebanking decisions. The question is whether there is more (or extra) information (in a given treebank) covered by the treebanking decisions in comparison with the state-of-the-art feature types, and if so then how that information can be extracted and exploited for addressing this problem. It might be possible that there are hidden correlated patterns of decision making (i.e. discriminant[3] choosing) inside human decisions. So, we would like to also know whether we can develop a method to learn such patterns, if there exists any, and use them for the reduction of the number of non-preferred parse trees of the parse forests (or simply, parse forest reduction).

Treebanking decisions are interesting for several reasons. These decisions record the fine-grained human judgements in the manual disambiguation process. Apart from these, these decisions allow to encode useful relationships between lexical words and the grammar rules of their distant grandparents without considering nodes (and hence, also grammar rules of those nodes) in the intermediate levels. State-of-the-art feature templates[4] such as local derivational configurations and active edges (will be discussed in Chapter 4) require exhaustive search in every node of the parse trees of the parse forests to generate their corresponding features. At one hand, this is a very time consuming process; on the other hand, only a small portion of the huge amount of these features are actually active and play informative role during disambiguation of the new sentences. If features can be extracted using treebanking decisions, then those features would be directly constrained by their corresponding treebanking decisions which would reduce the search space and also the total number of features significantly.

[3]Discriminants are the properties that hold for some analyses of a particular utterance but not for others. They will covered more elaborately in later Chapters.

[4]By *feature template*, we mean a generic or abstract definition for a certain type of features that share a common structure.

Another potential prospect is, if we can model human disambiguation process accurately by focusing on the human annotated decisions, then such a model will not only improve the performance of the parsing system, but can also be applied interactively in treebanking projects to achieve better annotation speed (e.g. by ranking the promising discriminants higher to help annotators make correct decisions).

In short, treebanking decisions have several potential characteristics which make them worth to examine.

1.3 Contribution of this thesis

This thesis presents an empirical study on the potentiality of the treebanking decisions as discriminative features for HPSG parse selection. Although, we mainly compare the features extracted using treebanking decisions and the state-of-the-art features, but the thesis also examines different types of decisions and their corresponding extracted features from different points of view. The contribution of the thesis can be summarised as following —

- We define feature templates specific to the treebanking decisions and show how they can used to extract potential features from parse trees.
- We presents comparative analyses among the features extracted using treebanking decisions and the state-of-the-art feature types. We show that, features extracted using treebanking decisions (henceforth Treebanking Decision Feature or simply *TDF*) are more informative, despite the total number of these features being much less than that of the traditional feature types.
- Our experimental results indicate TDFs are more robust than their traditional[5] counterparts on the out-of-domain data.
- We present analyses of the results and show that TDFs have higher number of active features which indicates that they are more efficient.
- We present analyses of the impact of different types of treebanking decisions (yes/no, annotated/inferred) on disambiguation accuracy gain.
- We present analyses of the performance of TDFs if they are used in a two stage re-ranker.

[5]We will use the trems *traditional* and *state-of-the-art feature* alternatively throughout this document.

- We develop a method to extract patterns of correlated discriminant from human decisions and to use them for parse forest reduction.
- Finally, we discuss the open questions that are raised from our study and include brief outline about further research on this topic.

1.4 Outline of the thesis

The remaining chapters of the thesis is organised as following —

Chapter 2 introduces the reader to the mathematical preliminaries of this thesis, defining entropy, maximum entropy models and maximum likelihood estimation.

Chapter 3 discusses the previous related works. It starts with the background on statistical parsing (especially for unification-based grammars). Then, it describes some discriminant-based treebanking environments. Finally, some recent directions in parsing research (such as parser adaptation, re-ranking) which we consider in some of our experiments are discussed.

Chapter 4 presents the features that provide the empirical basis for our experiments. We start with a brief discussion about the Redwoods-style treebanks that is used in this thesis. Following this, the sate-of-the-art feature types used are discussed. Then, we formally define treebanking decisions with examples and then include some additional reasons for our interest on these decisions. Finally, we present the feature templates, that we use for TDFs extraction, and draw distinctions between TDFs and the traditional features.

Chapter 5 describes the system that we develop for our experiments. We explain some of the main components of the system along with graphical presentation of their work flows. We also discuss how disambiguation model is trained and later used for testing.

Chapter 6 presents some of the key details related to the setup of our experiments. We describe various aspects of the experimental data and features. We also discuss the training models and evaluation measures.

Chapter 7 is dedicated to the description of the results of obtained from a series of experiments that analyze TDFs from different perspectives and illustrate their potentiality to be used for parse disambiguation.

Chapter 8 describes a method developed to extract patterns of correlated discriminants from human decisions. The chapter also describes the results when these patterns are used for parse forest reduction.

Finally, in Chapter 9 we summarize what we have learnt from this thesis. We conclude by giving suggestions for directions of future research.

Chapter 2

Mathematical Preliminaries

In this chapter we will review the mathematical foundations of the statistical models, called *maximum entropy* (MaxEnt) models, that are used for parse disambiguation in this thesis. A feature of maximum entropy (ME) modeling that makes it very attractive is that it is a general purpose technique that can be applied to a wide variety of problems in natural language processing (Berger et al., 1996; Ratnaparkhi, 1998; Johnson et al., 1999; Miles, 2000). However, it is not the purpose of this chapter to give a complete introduction, but rather to provide some background for the uninitiated reader. So, this chapter takes a quite general view on the models and only describes their mathematical foundations. Readers are advised to visit *Zhang Le's maxent page* [1] for more details.

We start with entropy, a key concept of information theory, in Section 2.1. This concept has traditionally played an important role in relation to the evaluation of language models, and also provides a natural transition to the framework of maximum entropy models, which is the topic of Section 2.2. Finally, in Section 2.3, we look into maximum likelihood estimation that is required to build conditional models.

2.1 Entropy

One of the key quantities in information theory, pioneered by Claude E. Shannon in the late 1940s, is entropy, which is a measure of the uncertainty associated with a random variable. To put differently, entropy is a measure of the average information content one is missing when one does not know the value of the random

[1] See http://homepages.inf.ed.ac.uk/lzhang10/maxent.html

variable (Shannon, 1948). For a random variable X, distributed according to a probability mass function p(x), its entropy is defined as:

$$H(X) = -\sum_{x \in X} p(x) \log p(x) \qquad (1)$$

Consider tossing a coin with known, not necessarily fair, probabilities of coming up heads or tails. The entropy of the unknown result of the next toss of the coin is maximized if the coin is fair (that is, if heads and tails both have equal probability 1/2). This is the situation of maximum uncertainty as it is most difficult to predict the outcome of the next toss; the result of each toss of the coin delivers a full 1 bit of information. However, if we know the coin is not fair, but comes up heads or tails with probabilities p and q, then there is less uncertainty. Every time it is tossed, one side is more likely to come up than the other. The reduced uncertainty is quantified in a lower entropy: on average each toss of the coin delivers less than a full 1 bit of information. The extreme case is that of a double-headed coin for which a tail never comes up. Then there is no uncertainty. The entropy is zero: each toss of the coin delivers no information.

So, the entropy is highest under the uniform distribution, i.e. when all outcomes are equally probable. One of the main concerns of information theory is how to best encode information. The entropy of a random variable provides a lower bound on the average number of bits needed to represent that variable (Cover and Thomas, 1991). With a non-uniform distribution, the coding scheme can take advantage of the fact that some events are more likely than others, and assign labels that require a smaller number of bits to encode these events, thereby reducing the average code length.

2.2 Maximum entropy models

Maximum entropy models offer a clean way to combine diverse pieces of contextual evidence in order to estimate the probability of a certain linguistic class occurring with a certain linguistic context. Contexts in NLP tasks usually include words, and the exact context depends on the nature of the task. For some tasks, a context consists of just a single word, while for others, it consists of several words and their associated syntactic labels and, sometimes, the grammatical rules used to assign those syntactic labels.

The principle of the maximum entropy models is: model all that is known and assume nothing about that which is unknown. In other words, given a collection

of facts, choose a model consistent with all the facts, but otherwise as uniform as possible (Berger et al., 1996; Ratnaparkhi, 1997). The MaxEnt principle is typically explained as an Occam's Razor argument for model selection, in the sense that we should never choose a model that is more complicated than necessary for explaining the empirical data.

MaxEnt models have been widely used for a range of tasks in NLP, including parse selection (Johnson et al., 1999; Miyao and Tsujii, 2002; Malouf and Van Noord, 2004). These models sometimes also go under other guises such as log-linear models, exponential models, Random Fields and Gibbs distributions. In many cases the differences between these models only pertain to the theoretical motivation rather than their practical implementation (Velldal, 2008). In this section, however, we focus on conditional maximum entropy models.

A MaxEnt model is given by a set of feature functions and corresponding weights. The specified feature functions describe properties of the data points, and the associated set of learned weights determines the contribution or importance of each feature. The real-valued features can describe arbitrary properties of the data points. For our purpose, we need to construct a probability distribution p over a set of parses X which are characterized by features $f_i(x)$ which may encode arbitrary characteristics of the parses. If we have a set of sentences W and a function $Y(w)$ that partitions X into the set of parses whose yield is $w \in W$, the conditional probability of parse x for sentence w is:

$$p(x|w;\theta) = \frac{exp(\sum_i \theta_i f_i(x))}{\sum_{y \in Y(w)} exp(\sum_i \theta_i f_i(y))} \qquad (2)$$

The value of $f_i(x)$ reflects the frequency of the i-th feature in a given parse x. The parameters θ_i, which is the weight of the corresponding i-th feature, can be estimated efficiently by maximizing the pseudo-likelihood of a training corpus (Johnson et al., 1999; Malouf, 2002):

$$L(\theta) = \sum_w \tilde{p}(w) \sum_{x \in Y(w)} \tilde{p}(x|w) \log p(x|w;\theta) \qquad (3)$$

The empirical probabilities $\tilde{p}(w)$ and $\tilde{p}(x|w)$ are derived from the training data.

However, to minimize over-fitting, a more effective approach is to use a penalized likelihood function for parameter estimation (Chen and Rosenfeld, 1999; Johnson et al., 1999). That means, rather than maximizing the likelihood (Equation (3)) to estimate the parameters θ_i, we instead maximize a penalized likelihood:

$$L'(\theta) = L(\theta) - \frac{1}{2\sigma^2}\sum_i {\theta_i}^2 \qquad (4)$$

This has the effect of imposing a Gaussian prior distribution on the parameter values with a mean of zero and a variance of σ^2, which in turn penalizes extreme feature values and tends to reduce over-fitting. The variance σ^2 is a smoothing parameter which sets the relative influence of the likelihood and prior: larger values of σ^2 results in less smoothing of the parameters θ_i.

An attractive property of the maximum entropy models is that, it is possible to integrate distinct but potentially overlapping sources of information, and features can be defined to take into account whatever aspects of the parse trees are considered important.

A potential drawback of the maximum entropy models is that Equation (2) requires access to all parses of a given corpus sentence, which is inefficient because a sentence can have an exponential number of parses. Two types of solution for this problem have been proposed. Geman and Johnson (2002), and Miyao and Tsujii (2002) present approaches where training data consists of parse forests (or feature forests), rather than sets of parses. Such approaches enforce strong locality requirements on features. Another type of solution (Miles, 2000) shows that it suffices to provide training with a representative sample of *Y(w)*.

2.3 Maximum likelihood estimation

Learning a MaxEnt model amounts to finding the values for the parameters θ_i that satisfy the constraints on expected feature values, and also uniquely determine the model with the highest entropy. It turns out that solving the constrained optimization problem of finding the model with the greatest entropy, is equivalent to solving the unconstrained optimization problem of finding the values of the parameters θ_i that maximizes the log-likelihood $L(\theta)$ of the training data X:

$$\hat{\theta} = \arg\max_{\theta} L(\theta) \qquad (4)$$

Thus, two different approaches – maximum likelihood and maximum entropy lead to the same solution. (Velldal, 2008)

Before we leave this chapter, for our purpose, we integrate a tool called the *Toolkit for Advanced Discriminative Modeling* (*TADM*) [2], which is based on the open-source estimate package by Malouf (2002), with our system (will be discussed in Chapter 5) for MaxEnt estimation. *TADM* is the implementation of the theories that we have discussed so far in this chapter.

[2] See http://tadm.sourceforge.net.

Chapter 3

Previous Related Works

Parsing have drawn a huge attention of both linguist and computer scientist community since the late 80's. The result in a massive number of scientific literatures that ranges from various grammatical frameworks to the variation of mathematical and statistical models. As mentioned before, the experiments done in this thesis examine a variety of approaches (such as log-linear models, re-ranking, parser adaptation, discriminant pattern extraction from human decisions, etc.) using features extracted from treebanking decisions. We review some of the notable previous studies done on those approaches which are closely related to our work. We begin in Section 3.1 with the recent developments of statistical parsing for unification-based grammars, especially using log-linear models. This is followed by some history on discriminant-based treebanking environments in Section 3.2. Finally, we briefly discuss in Section 3.3 about few of the works done on domain adaptation, re-ranking and self-training.

3.1 Statistical parse selection

As Velldal (2008) noted, stochastic elements are effective in parsing for two reasons: robustness and dealing with ambiguity. In many real-world applications, the input to a natural language analyzer can be expected to be noisy and erroneous. Moreover, there will always be cases where the grammar will fail to cover, no matter how broad-coverage the grammar has. A strict grammatical requirement will leave the parser brittle in the face of such input. Furthermore, often ungrammatical strings are comprehensible, and certain statistical parsers can assign an interpretation to such inputs.

The other issue is with ambiguity. The more the grammar has a rich and deep

coverage, the more is the chance that it may introduce greater number of syntactic analyses for structurally ambiguous sentences. A statistically guided parser can disambiguate such sentences by ranking the competing structures and selecting a single preferred analysis. There have been plenty of works on statistical parsing in last two decades, and historically, a major portion of them are focused on probabilistic context-free grammar (PCFG). So, in this section we will rather restrict ourselves only some of those works that are closely related to this thesis.

Abney (1997) show that, the methods used for defining weights of the features for a context-free grammar cannot be transferred to the more powerful family of unification-based grammars (UBGs) (such as HPSG and LFG, that can encode much richer syntactic and semantic constraints). The non-local dependencies created by the unification constraints, break the simple tree structures that are induced by the productions of a PCFG (Johnson et al., 1999). Also, simply computing relative frequencies does not generally result in a maximum likelihood estimate in the case of SUBGs. In order to account for the context-sensitive dependencies of UBGs, Abney (1997) show how a maximum likelihood distribution can be computed using log-linear models (see Section 2.2), with parameters estimated using a Monte Carlo-based gradient ascent procedure. The estimation procedure finds the optimal parameters of the weights that maximize the log-likelihood of the training corpus according to the model.

Abney (1997)'s concept was further refined by Johnson et al. (1999) to make parameter estimation more efficient to be practically feasible for grammars of realistic size. They suggest to use a so-called pseudo-likelihood estimator that finds parameters that maximize the conditional probabilities of the annotated parses given the strings in the training data. This approach also takes advantage of negative examples in the sense that the weights are chosen to maximize the probability of the preferred parses relative to the non-preferred ones for each string in the training corpus.

In recent years, several approaches have been developed using log-linear models for SUBGs, which are basically variants of the approaches of Abney (1997) and Johnson et al. (1999). With regard to this thesis, the most relevant one among them is the work of Toutanova et al. (2005) on training and comparing different conditional log-linear models for parse selection. Toutanova et al. (2005) describe experiments on HPSG parse disambiguation using the 1st growth of LinGO Redwoods HPSG treebank [1] which provides much richer representation than Penn Treebank that has

[1] A Redwoods-style treebank is different from other treebanks in that the treebank itself changes as the ERG is improved.

been the focus in parsing research for many years. The LinGO Redwoods HPSG treebank (see Section 4.1 for more details) is annotated with the HPSG analysis licensed by the LinGO English Resource Grammar (ERG; (Flickinger, 2000)). The HPSG signs are typed feature structures that encode fine-grained syntactic information, and also include a logical-form meaning representation based on Minimal Recursion Semantics (MRS; (Copestake et al., 2006)). The authors compare performance of generative and discriminative models on disambiguation task using analogous features over derivation trees. The first generative model is a PCFG model with production features defined for the rule schemata in the derivation trees of the HPSG analyses in the treebank. The second model is also a PCFG model in which production features are extended to include ancestor information for up to a maximum of four dominating nodes. The authors also train similar PCFG models using more conventional phrase structure trees derived from the derivation trees. They observe that the disambiguation accuracy for these models (based on phrase structure trees) is lower than that for the models trained on derivation tree representations of the Redwoods. Another PCFG-style model is trained on semantic dependency trees that are extracted from these MRS representations. They also implement a conventional trigram HMM tagger that defines a joint probability distribution over pre-terminal tag sequences and yields of the derivation trees. Finally, several of the generative models are combined in a single model by linear interpolation. After that, the authors train corresponding discriminative models that are defined using exactly the same feature sets, but are trained to maximize the conditional log-likelihood of the treebank data. For all the different model configurations, the discriminative variants substantially outperform their generative counterparts, resulting in relative reductions in error rate of up to 28%. The best model among them reach up to 82.5% exact match accuracy.

The study of Toutanova et al. (2005) resulted in some important observation. They show that production features defined over derivation trees give better results than with standard phrase structure trees. They also show extending the feature set to include ancestor information provides much better accuracy. However, the authors observe that, the information in the semantic dependency trees do not seem to contribute significantly to disambiguation. (Velldal, 2008)

Osborne and Baldridge (2004) propose ensemble-based active learning for parse selection. The authors experiment on 3rd growth of Redwood treebank. Three distinct feature sets – configurational, n-gram, and conglomerate are used to train log-linear models. The authors argue that these feature sets incorporate different aspects of the parse selection task and have different properties. The configurational

feature set is the derivation tree features, also known as local configuration features (see Section 4.2), described by Toutanova et al. (2003) that take into account ancestor relationships among the nodes of the trees. The n-gram set, described by Osborne and Baldridge (2004), also uses derivation trees; however, it uses a linearized representation of trees to create n-grams over the tree nodes. According to the authors, this feature creation strategy encodes many (but not all) of the relationships in the configurational set, and also captures some additional long-distance relationships. The conglomerate feature set uses a mixture of features gleaned from phrase structures, MRS structures, and elementary dependency graphs. Each of these representations contains less information than that provided by derivation trees, but together they provide a different and comprehensive view on the ERG semantic analyses. The features contributed by phrase structures are simply n-grams of the kind described above for derivation trees. The features drawn from the MRS structures and elementary dependency graphs capture various dominance and co-occurrence relationships between nodes in the structures, as well as some global characteristics such as how many predictions and nodes they contain. The authors report that an ensemble of the three parse selection models (based on the different feature sets) achieves a 10.8% reduction in error rate over the best single model. Their best result achieves a 73% reduction in annotation cost compared with single-model random sampling and a 77% exact match accuracy. [2]

Crysmann (2008)'s experiments are conducted on German subset of Verbmobil corpus. The author uses GG, a large-scale HPSG grammar for German and report 81.49% exact match accuracy. The results indicate that, for German (which has a less rigid word order than English), when grand parenting is used, n-gram does not contribute any significant improvement in accuracy. The results also indicate active edges deteriorate accuracy if grandparenting is already available.

Some of the other related works include estimating contribution of language model in comparison with conditional models in parse selection and realization by Velldal (2008), partial parse selection by Zhang et al. (2007), log-linear model used for Dutch by Malouf and Van Noord (2004), etc. Various results are report in CoNLL 2007 shared task on dependency parsing where the best system obtained score of 89.61 on the WSJ section of Penn treebank (Nivre et al., 2007). Riezler et al. (2002) report parsing WSJ treebank using LFG and discriminative estimation techniques and their result reached 79% F-score.

[2] It should be noted, growth 1 of Redwood (upon which Toutanova et al. (2002) result is reported) has considerably less ambiguity than growth 3.

3.2 Discriminant-based treebanking environments

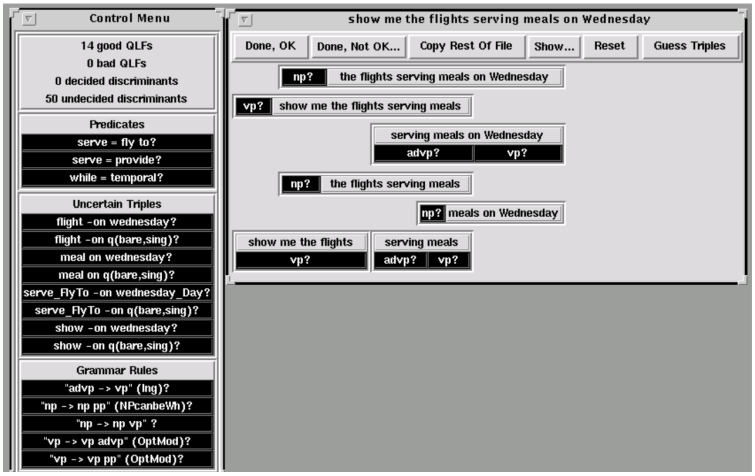


Figure 3.1: The SRI Cambridge Treebanker GUI.

Carter (1997) present a graphical tool for treebanking named as *the SRI Cambridge Treebanker* (or simply, *the Treebanker*). Primarily, the tool was developed as a component for a spoken language translator system. It presents a user, who need not be a system expert, with a range of properties that distinguish competing analyses for an utterance and that are relatively easy to judge. This allows training on a corpus to be completed in far less time, and with far less expertise, than would be needed if analyses are inspected directly.

Given an input string, zero or more quasi-logical form (QLF; (Alshawi, 1990)) analyses of it is created by applying unification-based syntactic rules and their corresponding semantic rules. Then, various properties are extracted from those QLFs and presented to non-expert users in a form that they can easily understand. Carter (1997) name the properties that hold for some analyses of a particular utterance but not for others as discriminants. The tool is developed based on the assumption that, discriminants that fairly consistently hold for correct but not (some) incorrect analyses, or vice versa, are likely to be useful in distinguishing correct from incorrect analyses at run time. Thus for training on an utterance to be effective, one needs to provide enough "user-friendly" discriminants to allow the user to select the correct analyses, and as many as possible "system-friendly" discriminants that, over the corpus as a whole, distinguish reliably between correct and incorrect analyses. The data collected from the training the system by the user is then used by the

speech recognizer. Figure 3.1 shows the graphical form for the sentence "Show me the flights to Boston serving meal on Wednesday" that is presented to the user.

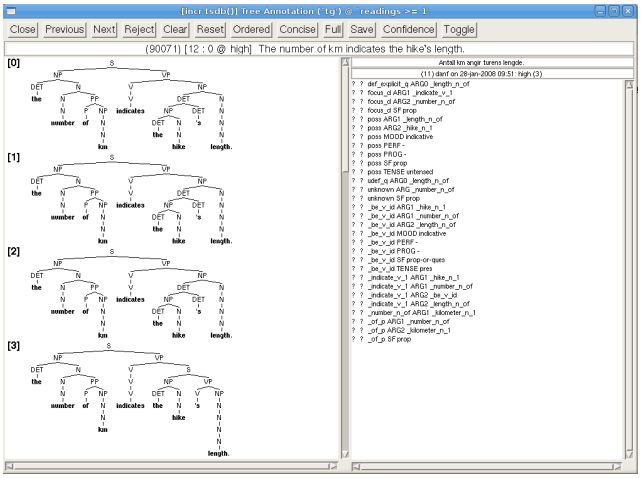


Figure 3.2: `[incr tsdb()]`treebank annotation GUI.

While *the SRI Cambridge Treebanker* is basically one of the earliest (if not the first) discrimination based treebanking environments, it operates with much simpler grammar than deep grammars, and hence, does not deal with huge number analyses that a rich deep grammar can produce. `[incr tsdb()]` serves this purpose (Oepen, 2001). `[incr tsdb()]` competence and performance profiler, a tool for grammar and system profiling and treebanking, have been developed, maintained and enhanced by Stephan Oepen and other contributors for more than a decade. The term *grammar profiling* refers to a methodology that builds on structured and annotated collections of test and reference data (traditionally known as *test suits*) (Oepen and Flickinger, 1998). A competence and performance profile gives a rich, precise, and structured snapshot of the system behaviour at a given development point. These profiles are stored in a relational database that accumulates a precise record of system evolution that later serves the basis for various report generation and data analysis via descriptive statistical measures. To create test suits, `[incr tsdb()]` provides a graphical user interface (GUI) to facilitate treebanking annotation. `[incr tsdb()]` supports the Redwoods-style treebanking (will be discussed later in Section 4.1). The annotator selects the correct analysis (or, occasionally, rejects all analyses). Selection is done through the choices of discriminants. Like *the SRI Treebanker*, the system selects features that distinguish between different parses, and the annotator selects or rejects the features until only one parse remains. The number of decisions for

each sentence is normally around log_2 *(x)* of the number of parses, although sometimes a single decision can reduce the number of remaining parses by more or less than half. For each action (selection or rejection of discriminant) taken by the annotator, the system accumulates the state of the discriminants, in all the analyses of the corresponding sentence, which either agree or conflict with the action of the annotator, and automatically marks them as postive (selected/not conflicting) and negative (rejected/conflicting) discriminants accordingly. All the actions taken by both the annotator and system are saved in log/database file.

A key aspect of the Redwoods approach to treebanking is all linguistic information in the annotations is grounded in the external HPSG grammar (Oepen et al., 2002). This can be contrasted with approaches where the treebank annotations are themselves taken to implicitly define a grammar. By instead anchoring the annotations to an external grammar, the internal consistency of the treebank is guaranteed, It also makes the resource more dynamic, in that annotations can easily be updated to reflect revisions in the grammar as it develops and improves over time comparing with the pre-recorded previous actions of the annotators (Velldal, 2008). The implementation of the all the procedures that go into developing treebanks (such as the procedures for synchronizing a treebank with a given grammar version, the generation of paraphrases when symmetrizing a parse treebank, the labeling procedures, and so forth), is based on the tight integration between `[incr tsdb()]` and the open-source grammar engineering system *Linguistic Knowledge Builder*[3] (LKB; (Copestake, 2002)). Figure 3.2 shows the treebank annotation GUI of `[incr tsdb()]`.

3.3 Domain adaptation, re-ranking and self-training

Parse re-ranking is a recent direction in parsing research. Parse re-ranking can be viewed as a two-stage process (Charniak and Johnson, 2005). In the first stage a parser is asked to create an initial list of possible parses for a given utterance. Usually, this list consists of the k-best parse trees. In the second stage, a re-ranker is applied to the list of parse suggestions, in order to select a single most likely tree. Another recent research direction, known as parser self-training, is to take an existing parser, parse extra data and then create a second parser by treating the extra data as further training data (McClosky et al., 2006). McClosky and Charniak (2008) apply this technique to *parser adaptation*[4]. The authors self-train the Charniak/Johnson

[3] See http://wiki.delph-in.net/moin/LkbTop for details about LKB.

[4] *Parser adaptation* attempts to leverage existing labeled data from one domain and create a parser capable of parsing a different domain. (McClosky et al., 2006)

Penn-Treebank parser (Charniak and Johnson, 2005) using unlabeled biomedical abstracts. They weight the original WSJ hand annotated sentences equally with self-trained Medline data before testing on a corpus of hand-parsed sentences from the Genia Treebank (Tateisi et al., 2005).

One of the first attempt for parser adaptation based on unification-based grammars was by Hara et al. (2005) (also see Hara and Tsujii (2007)). The authors develope a log-linear model with additional features on GENIA treebank (Kim et al., 2003) and use it with original model of HPSG parser built on Penn treebank. After re-training the combined model, the achieved F-score on the biomedical domain (86.87) by the parser is close to that of the original parser on Penn Treebank (87.16). The original parser, Enju (Miyao and Tsujii, 2005), represents disambiguation model in a packed forest structure. Hara et al. (2005)'s parser exploits that packed structure to add additional features on conjunctive nodes.

Chapter 4

Treebanking Decisions and Features

In this chapter we will be taking a detail look at the features that provide the empirical basis for our experiments. Generally speaking, a treebank is a data resource where strings have been annotated with grammatical structure, typically in the form of parse trees. With regard to Redwoods-style treebank, annotation does not consist of drawing parse trees; instead involves picking the correct parse tree out those produced by the parser. In doing so, human annotators make judgements about the discriminative grammar rules that differentiate the parse trees from each other. We call these judgements (annotated) *Treebanking Decisions*. We will see how we can extract relevant features from the derivation trees using these treebanking decisions and also how to take on account other aspects (such as context) which are not directly observed in the treebanking decisions despite those aspects would have played vital role on decision-making process of the human annotators. However, we will postpone the presentation of discriminative model building using these features until Section 6, and reserve the current chapter to explain how these features are different from the state-of-the-art features. We start this chapter with a brief discussion about the Redwoods-style treebanks and the resources they rely on. After this, the sate-of-the-art feature types used for these treebanks is discussed in Section 4.2. In Section 4.3, we formally define treebanking decisions with examples. Following this in Section 4.4, we express more motivations for our interest on these decisions. Section 4.5 describes the feature templates that we use for extracting features. Finally, Section 4.6 concludes the chapter by drawing distinctions between treebanking decision features and traditional features.

4.1 The Redwoods-style treebanks

The Redwoods-style treebanks[1] is a family of treebanks that share the same basic methodology and the same underlying grammar (Oepen et al., 2002; Velldal, 2008). They have been annotated in accordance with an existing hand-crafted grammar. More concretely, each string in the Redwoods corpora is annotated with an HPSG analysis assigned by the LinGO English Resource Grammar (ERG; (Flickinger, 2000)). All linguistic information in the annotations is grounded in the external grammar, and this is a key aspect of the Redwoods approach to treebanking (Oepen et al., 2002). This can be contrasted with approaches where the treebank annotations are themselves taken to implicitly define a grammar. By instead anchoring the annotations to an external grammar, the internal consistency of the treebank is guaranteed. It also makes the resource more dynamic, in that annotations can easily be updated to reflect revisions in the grammar as it develops and improves over time (Velldal, 2008). Oepen et al. (2002) describe a method for semi-automatically updating and maintaining treebanks with respect to the changes in the grammar, based on the notion of elementary discriminants (Carter, 1997). These discriminants correspond to the basic, differentiating properties of local ambiguities in the parse forest. By toggling the activation of these markers, the annotator can usually disambiguate a parsed string in very few steps (we have already discussed this in Section 3.2). The implementation of all the procedures that go into developing treebanks (such as the procedures for synchronizing a treebank with a given grammar version, the generation of paraphrases when symmetrizing a parse treebank, the labelling procedures, and so forth), is based on the tight integration of `[incr tsdb()]` (Oepen, 2001) and the LKB system (Copestake, 2002). The LKB system is a grammar and lexicon development environment for use with unification-based linguistic formalisms; while the `[incr tsdb()]` environment is a tool for grammar and system profiling and treebanking.[2] Together these systems provide an extensive software suite for grammar engineering and profiling.

The fundamental data type of the Redwoods treebanks is the derivation tree. The internal nodes of these trees correspond to identifiers of rules in the underlying grammar, such as the head-complement or head-adjunct schema, while the preterminal yields correspond to identifiers of the lexical entries. These derivation tree representations form the basis for the extraction of the state-of-the-art features to

[1] See http://redwoods.stanford.edu/ for further information.

[2] It is out of the scope of this thesis to describe these systems in details, although we mention briefly relevant properties of the systems whenever it is required for better understanding (e.g. Section 3.2 contains some discussion on `[incr tsdb()]`). Readers are advised to refer to the corresponding references for details.

build discriminative models. Figure 4.1 shows an example of derivation tree, where the pre-terminal nodes have been mapped to the corresponding abstract lexical types. One thing to notice from the example is, the tree is maximally binary. This is because, the HPSG rules inside ERG do not allow the trees to have more than two branches.

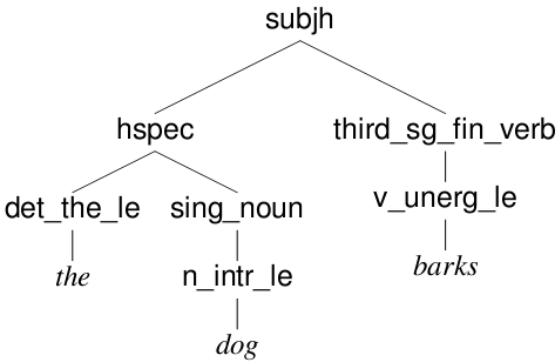


Figure 4.1: Example of HPSG derivation tree. Phrasal nodes are labeled with identifiers of grammar rules, and (pre-terminal) lexical nodes with class names for types of lexical entries. Although, the original derivation tree format has pre-terminals corresponding to lexical identifiers, this figure shows a modified format where these identifiers are mapped to one of the abstract lexical types of ERG.

4.2 The state-of-the-art feature types

The main three state-of-the-art feature types defined over HPSG derivation trees are — local configurations, active edges and n-grams. *Local configurations* are the local sub-trees. While they encode the full sequence of daughters in the local sub-trees, *active edges* record only one of the daughters in turn which allows to reduce the effects of data sparseness. Both of these feature types are extended using various degrees of grandparenting (i.e. ancestor information). Feature type *n-grams* record n-grams of lexical types, extracted from the pre-terminal of the derivation trees. Features extracted using this feature type are sometimes extended using lexicalization (i.e. addition of surface tokens). The optimal feature set size for all of these feature types are determined empirically based on some parameters such as optimal level of grandparenting, various frequency cut-offs, etc. (Velldal, 2008)

Table 4.1 shows some sample features extracted from the tree in Figure 4.1 using these feature types.

Feature type	Extension	Sample feature
local configuration	–	subjh hspec third_sg_fin_verb
local configuration	–	hspec det_the_le sing_noun
local configuration	grandparenting level = 1	subjh hspec det_the_le sing_noun
n-gram	n = 1	n_intr_le
n-gram	n = 2	det_the_le n_intr_le
n-gram	n = 1, lexicalized = Yes	n_intr_le dog
active edges	–	subjh third_sg_fin_verb
active edges	–	subjh hspec

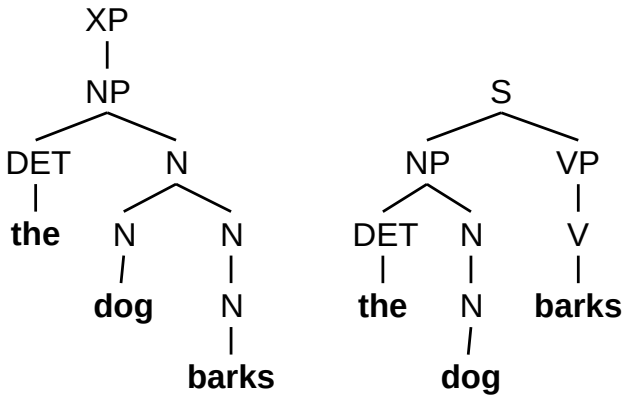
Table 4.1: Example of the state-of-the-art features extracted from the derivation tree in Figure 4.1.

4.3 Treebanking decisions

One of the defining characteristics of Redwoods-style treebanks is, treebank annotation does not consist of drawing parse trees. Instead, the candidate trees are constructed automatically by the grammar, and then manually disambiguated by human annotators. In doing so, linguistically rich annotation is built efficiently with minimum manual labor. In order to further improve the manual disambiguation efficiency, systems like `[incr tsdb()]` computes the difference between candidate analyses. Instead of looking at the huge parse forest, the treebank annotators selects or rejects the features that distinguish between different parses, until only one parse remains. The number of decisions for each sentence is normally around $log_2(n)$ where n is the total number of candidate trees. As discussed in Chapter 3, a similar method is also proposed in (Carter, 1997).

Formally, a feature that distinguishes between different parses is called a **discriminant**. For Redwoods-style treebanks, this is usually extracted from the syntactic derivation tree of the HPSG analyses. Figure 4.2 shows a set of example discriminants based on the two candidate trees.

A choice (acceptance or rejection, either manually annotated or inferred by the system) made on a discriminant is called a **decision**. In the above example, suppose the annotator decide to accept the binary structure *the dog* || *barks* as a subject-head construction and assign value **yes** to discriminant **D1**, the remaining discriminants



D1	SUBJH	the dog || barks
D2	HSPEC	the || dog barks
D3	FRAG_NP	the dog barks
D4	HSPEC	the || dog
D5	NOUN_N_CMPND	dog || barks
	...	...
D6	PLUR_NOUN_ORULE	barks
D7	*v_-_le*	barks
D8	*n_-_mc_le*	barks

Figure 4.2: Example forest and discriminants

will also receive inferred values by deduction (**no** for **D2**, **no** for **D3**, **yes** for **D4**, etc.). These decisions are stored and used for dynamic evoluion of the treebank along with the grammar development.

4.4 Why treebanking decisions

We have already mentioned some key characterisitcs of treebanking decisions in Section 1.2. This section provides additional reasons for our interest on treebanking decisions.

As we pointed out before, treebanking decisions record fine-grained human judgements in the manual disambiguation process. This is different from the tranditional use of treebanks to build parse selection models, where a marked gold tree is picked from the parse forest without concerning detailed selection steps. Recent study on double annotated treebanks (Kordoni and Zhang, 2009) shows that annotators tend to start with the decisions with the most certainty, and delay the "hard" decisions as much as possible. As the decision process goes, many of the "hard" discriminants will receive inferred value from the certain decisions. This

greedy approach helps to guarantee high inter-annotator agreement. Concerning the statistical parse selection models, the discriminative nature of these treebanking decisions suggests that they are highly effective features, and if properly used, they will contribute to a efficient disambiguation model.

4.5 Feature extraction using treebanking decisions

We use three feature templates for the treebanking decisions for feature extraction. We refer to the features extracted using these templates as **TDF** (Treebanking Decision Feature). The feature templates are —

- **T1**: *discriminant + the abstract lexical types of the lexical words that it covers*;
- **T2**: *discriminant + rule(immediate left child)*[3] *+ rule(immediate right child)*; and
- **T3**: *instances of T2 + rule(parent) + rule(sibling).*

T1 re-presents the original decision; while T2 and T3 encode the context where type 1 is observed. *TDF*s of T1, T2 and T3 are combinedly referred as **TDFC** or TDF*s with context*. Table 4.2 shows some examples of *TDF*s extracted using the above mentioned templates. In Section 6.2, some more detail about the use of treebanking decisions for feature extraction is explained.

Feature template	Treebanking Decision	Sample TDF
T1	D4 **hspec** the || dog	hspec + *det_the_le* + *n_intr_le*
T2	D4 **hspec** the || dog	hspec + *det_the_le* + *sing_noun*
T3	D4 **hspec** the || dog	hspec + *det_the_le* + *sing_noun* + *subjh* + *third_sg_fin_verb*

Table 4.2: Example of the *TDF*s extracted from the derivation tree in Figure 4.1 using the treebanking decision *"D4 **hspec** the || dog"* in Figure 4.2 .

4.6 How are *TDF*s different from the traditional features?

There are basically two ways *TDF*s that differ from traditional features — the way they are structured and the way they are extracted. Consider the tree in

[3] *rule(X)* represents the HPSG rule, applied on *X*, extracted from the corresponding derivation tree.

Figure 4.1. None of the feature types mentioned in Section 4.2 can relate two (or more) internal nodes in a single feature excluding other nodes which appear in between them. So, for example, the traditional features cannot put *subjh* and lexical entry types (i.e. *det_the_le*, *n_intr_le* and *v_unerg_le)* in one feature without considering intermediate nodes such as *hspec* (see Figure 4.1). But if we use decision *D1* in Figure 4.2 and feature template *T1* (described in Section 4.5), it is possible to record such TDF. In other words, TDF allows to omit certain details inside the features by encoding useful relationships between lexical types of the words and their distant grandparents without considering nodes in the intermediate levels. This provides somewhat underspecification for the arbitrary HPSG rules applied in the intermediate nodes. This encoding is consistent with the original decision taken by the human annotators, because human annotators also do not see the details of internal nodes in the GUI of `[incr tsdb()]`. Instead they see the *discriminants* and the words on which those discriminants are applied as certain construction structures. However, human annotators do consider contexts of the discriminants before accepting/rejecting, and these contexts include both ancestor and siblings. These kind of information is recorded in *TDF*s by templates *T2* and *T3*. Though, state-of-the-art feature types encode ancestor information (through grandparenting), they do not consider siblings of the distant grandparents (in our case, discriminants) of the lexical entries that form their contexts. Thus, in these two ways *TDF*s structurally differ from the state-of-the-art features.

The other distinction among them is how they are extracted. In case of the state-of-the-art feature types, the search space is all the possible matches (which is huge) for the respective feature types. Among these matches, *'relevant'* features are finally kept for later processing. A feature is *'relevant'* for a particular sentence, if there are (at least) two readings of that sentence and inside the derivation trees of those readings the frequencies (i.e. number of times observed) of this feature are not equal. With regard to the *TDF*s, there are no notion of *'relevance'*. All the *TDF*s extracted are valid. In stead of generating all possible matches and then pruning, the selections of the *TDF*s are directly restricted by the treebanking decisions themselves. That means, for a particular treebanking decisions and a derivation tree, the original sentence of that derivation tree must contain the words covered by the decision, and also the discriminant of the decision must be present in that derivation tree before searching and selecting appropriate *TDF*s for that decision.

It should be noted that, we do not use treebanking decisions made for the parse trees of one sentence to extract features from the parse trees of another sentence.

To put differently, each set of treebanking decisions per sentence is only used for the parse trees of that particular sentences. Hence, the *TDF*s are highly correlated to the corresponding constructions and corresponding sentence from where they are extracted. The state-of-the-art are not exclusive to certain types of sentences, rather they are in general features for the whole dataset.

Chapter 5

System Overview

In this chapter, we will discuss the system[1] that we develop to carry out the experiments. However, we will avoid inclusion of finer implementation details, and rather focus on describing some of the main components of the system from abstract level. We use the system, mainly, for the disambiguation experiments using treebanking decisions. Nonetheless, the system is also capable of analysing the results from different perspectives and giving statistics that allow to observe different characteristics of the disambiguation model which cannot be expressed with only the percentage of accuracy.

We start first describing the feature extractor component in Section 5.1. Readers are recommended to look at the graphical explanation (Figure 5.0) for better understanding of the process. In Section 5.2, we discuss how disambiguation model is trained and later used for testing. For the theoretical background of the disambiguation model, we encourage readers to refer to the Chapter 2. Section 5.3 is dedicated to some performance analysis tasks, other than accuracy calculation, done by the system. Following this, in Section 5.4 and 5.5, we talk about the components used for re-ranking and parse forest reduction experiments. For calculating disambiguation accuracy for the models built using the state-of-the-art features, we use a system called *LOGON* (Oepen et al., 2004). We conclude the chapter by briefly describing how this system is used.

There is a component of our system for extracting patterns of correlated discriminants from human annotated decisions. We will discuss it separately in Chapter 8.

[1]The system has been developed using Java and it is integrated with a third-party component called *TADM* (more in Section 5.2) built using C++.

5.1 Feature extractor

The process of feature extraction is described by the Algorithm 5.1. As shown in Line *3*, only those decisions are used for a certain experiment whose types (*'annotated Yes'/'annotated No'/'inferred Yes'/'inferred No'*) are under consideration for that experiment. The system consider the parse trees of a sentence if the current decision corresponds to the same sentence (Line *4*). To put differently, decisions taken for one sentence is not used for feature extraction for derivation trees of another sentence. Features are extracted (using templates T1, T2 and T3 described in Section 4.5) for a decision from a sub-tree of a parse tree only if the corresponding discriminant (of that decision) matches with the label (i.e. HPSG rule) of a node of that sub-tree and the terminal nodes of the branches from that node have exactly the same words (in same order) like the words inside the decision (Line *6–8*). Figure 5.0 shows graphically how this is done.

Algorithm 5.1: Feature extraction

1: *featureSet* := $\emptyset$
2: **for** each *deci* $\in$ *treebankingdecisions* **do**
3: **if** *deci* $\rightarrow$ *type* $\in$ *decisionTypesToConsiderForFeatureExtraction* **then**
4: *derivationTreesList* := List of the parse trees of the sentence having same ID as *deci*$\rightarrow$*sentenceID*
5: **for** each *dTree* $\in$ *derivationTreesList* **do**
6: *dNodesList* := List of all the nodes of *dTree* whose labels are identical as *deci*$\rightarrow$*discriminant*
7: **for** each *node* $\in$ *dNodesList* **do**
8: **if** *node*$\rightarrow$*leftBranch*$\rightarrow$*allLexicalStrings* $\equiv$ *deci*$\rightarrow$*left*$\rightarrow$*allLexicalStrings* AND *node*$\rightarrow$*rightBranch*$\rightarrow$*allLexicalStrings* $\equiv$ *deci*$\rightarrow$*right*$\rightarrow$*allLexicalStrings* **then**
9: Create features for *deci*$\rightarrow$*discriminant* using the templates *T1, T2* and *T3*.
10: Add the features in *featureSet* if they do not already exists.
11: **end if**
12: **end for**
13: **end for**
14: **end if**

15: **end for**
16: **return** *featureSet*

5.2 Component for training maximum entropy models and testing

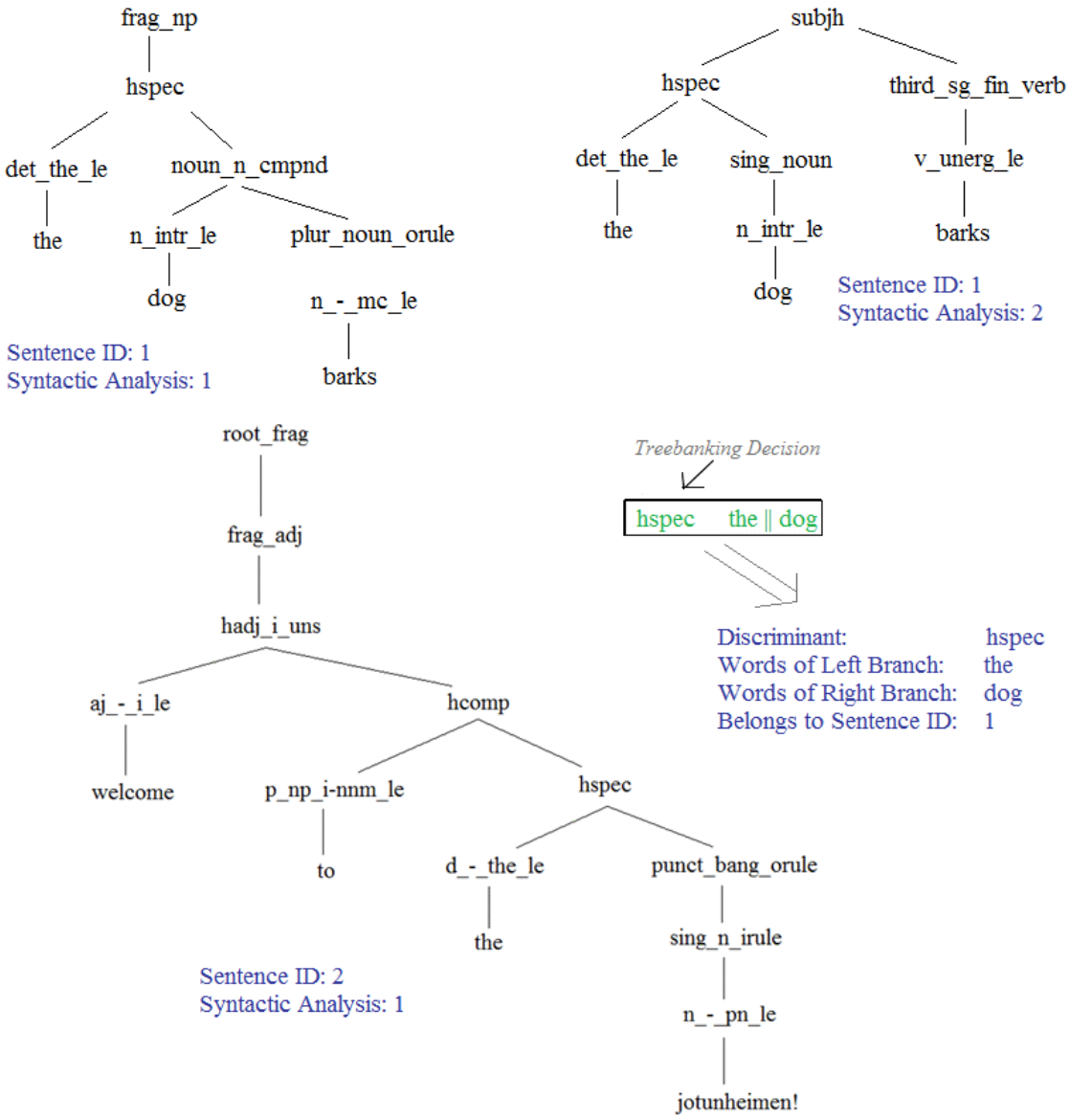
Our system is integrated with the *Toolkit for Advanced Discriminative Modeling (*TADM*)* [2], which is based on the open-source estimate package by Rob Malouf (Malouf, 2002), for MaxEnt esitmation. *TADM*, in turn, uses open-source software libraries such as the *Portable Extensible Toolkit for Scientific Computation (PETSc)*[3] and the *Toolkit for Advanced Optimization (TAO)*[4] for numerical optimization.

TADM accepts event files as input and returns the discriminative model in the form of real-valued weights of the features after training. Example of an event file is as following:

```
2
1.0 2 0 1 1 2
0.0 2 0 3 2 1
3
1.0 1 3 1
0.0 2 0 2 2 2
0.0 1 2 1
```

The first part of a event file is a header (not shown in the above example), bracketed by lines containing "&*header*" and "/". The header is optional and, if present, is ignored. The remaining of the file consists of one or more blocks. The first line of each block is the number of events (i.e. parse trees) for the corresponding context (i.e. sentence). The lines containing 2 and 3 in the above example denote the number of events for their corresponding context. The next lines contain events (i.e parse trees). Each event line has a so-called event frequency which is either 1.0 (preferred) or 0.0 (dis-preferred) for parse selection model. The event frequency is followed by the number of feature-value pairs, and then the corresponding pairs of feature IDs and values (i.e. number of times a particular feature has been observed inside the corresponding parse tree). Feature IDs are numbered starting with zero.

[2] See http://tadm.sourceforge.net.
[3] See http://www.mcs.anl.gov/petsc.
[4] See http://www.mcs.anl.gov/tao.


frag_np
hspec
det_the_le
noun_n_cmpnd
the
n_intr_le
plur_noun_orule
dog
n_-_mc_le
barks
Sentence ID: 1
Syntactic Analysis: 1
subjh
hspec
third_sg_fin_verb
det_the_le
sing_noun
v_unerg_le
the
n_intr_le
barks
dog
Sentence ID: 1
Syntactic Analysis: 2
root_frag
frag_adj
hadj_i_uns
aj_-_i_le
hcomp
welcome
p_np_i-nnm_le
hspec
to
d_-_the_le
punct_bang_orule
the
sing_n_irule
n_-_pn_le
jotunheimen!
Sentence ID: 2
Syntactic Analysis: 1
Treebanking Decision
hspec the || dog
Discriminant: hspec
Words of Left Branch: the
Words of Right Branch: dog
Belongs to Sentence ID: 1

(a)

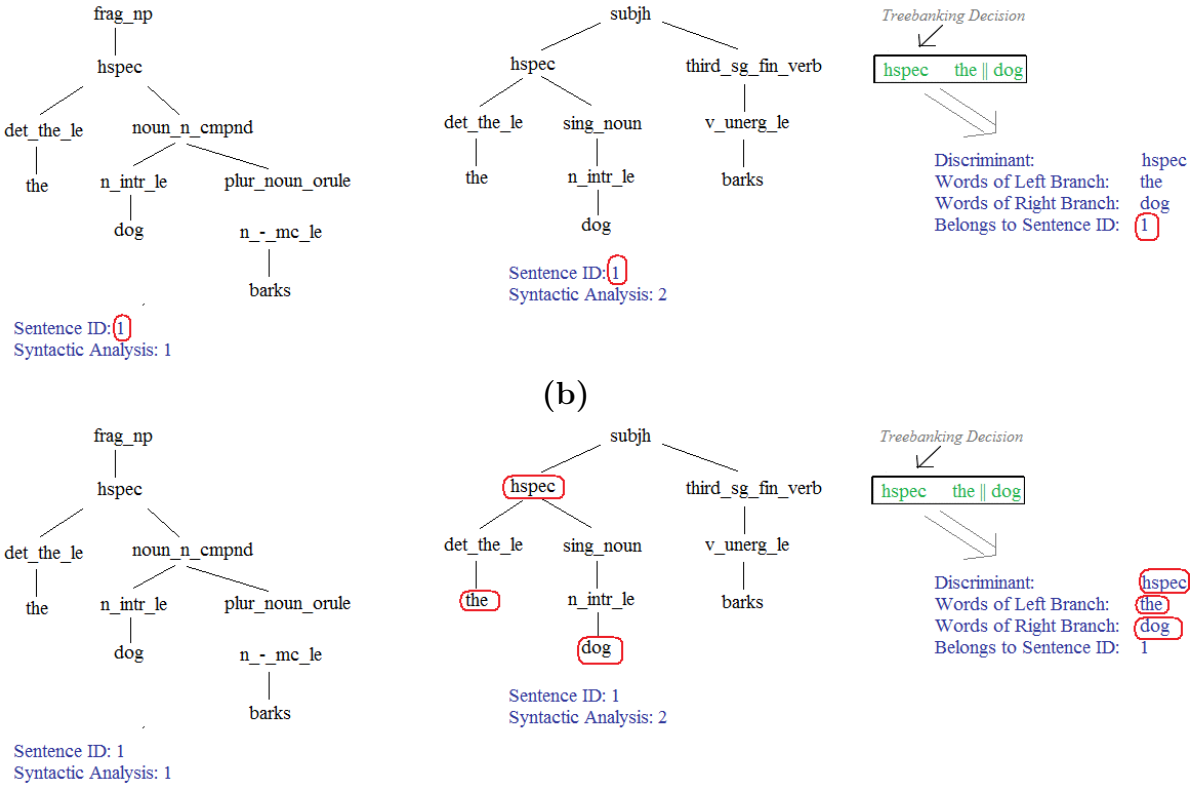


Figure 5.0: Steps of feature extraction: *(a)* sample parse trees and treebanking decision, *(b)* corresponding parse trees for the decision (i.e. belonging to the same sentence) is selected, *(c)* presence of the elements of the treebanking decision is identified in one of tree, *(d)* features are extracted using templates.

Each feature ID can appear only once in an event line, and must have a value greater than zero. Event lines with a zero event frequency are ignored for computing the entropy. Any feature with an expected value of zero is ignored (i.e. the corresponding parameter is set to 0.0).

Our system traverses the parse trees of those sentences for whom there is at least one preferred analyses selected by the human annotators, and write entries for them in the event file. For the other input parameters of *TADM*, we use the default values.

Like the training items, the system also encodes the test sentences as event file before passing to *TADM*. The returned output is the ranking of the parse trees of the sentences along with their scores. This scoring is done based on the weights of the features. Once the ranked parse trees are available, the outcomes are matched with the gold trees (i.e. preferred by human annotators) of the sentences and the *exact-match* accuracy and *5-best* accuracy is computed (see Section 6.5).

5.3 Performance analyser

The system analyzes the performance of the disambiguation model of a particular feature set (see Section 6.3) not only using the accuracy but also based some other aspects of the results. These analyses are done by the performance analyser component of the system. The component performs the following tasks —

- Calculate *Active Features*, *Feature Type Hit Count (FTHC)* and *Feature Hit Count* (see details in Section 6.5).
- Track accuracy variation with respect to the sentence length.
- Provide various statistics regarding feature extraction (such as which type of decisions yield how many features and so on).

5.4 Re-ranker

The re-ranker component of the system takes as input top n ranked parse trees per sentence. The initial ranking is done using the disambiguation model of local configuration feature set (see Section 6.3). The re-ranker uses the disambiguation model of TDFC feature set (see Section 6.3) and provide as output the new ranking of these n parse trees as output. After that, accuracy based on these new ranked items are calculated. Figure 5.2 shows the re-ranking process.

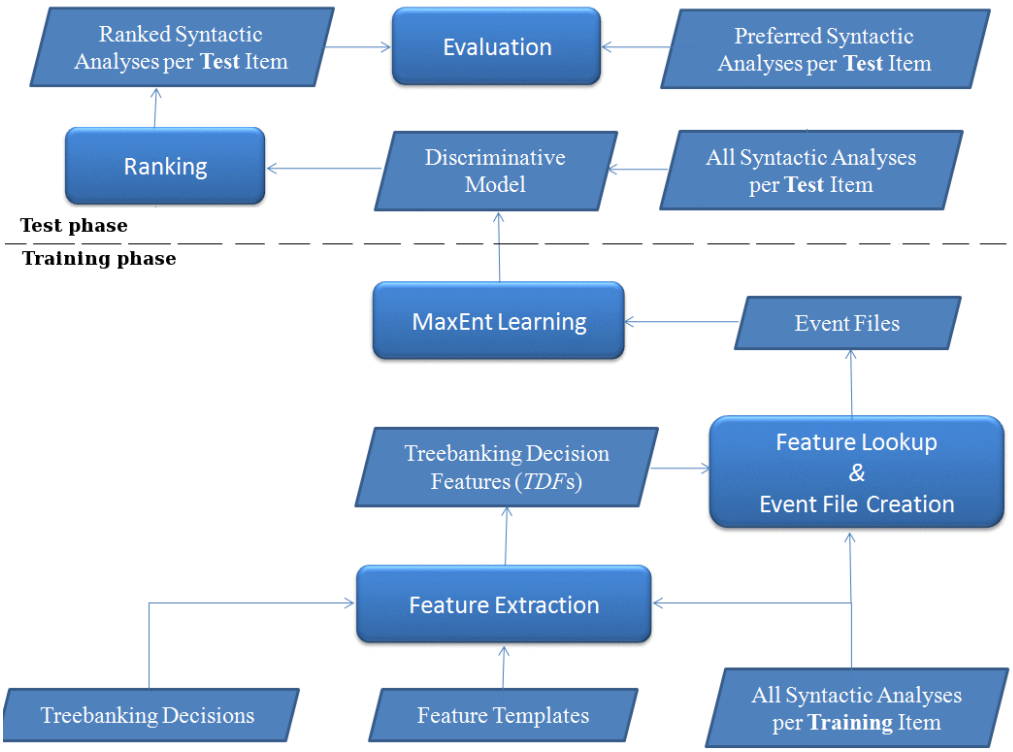


Figure 5.1: Work flow of the whole process of parse disambiguation with treebanking decisions.

5.5 Component for parse forest reduction using ranked *TDF*s

This particular component takes as input the weights of the TDFs calculated by the *TADM* and rank them according to their weights. For the items which have more than M readings, the ranked TDFs are used (in the order of their ranking) to split the parse forest into relevant and irrelevant parts. Then, the relevant part is again taken on consideration and splitted into two parts. The process continues until either there is no more such split is possible, or the number of times a particular parse forest is reduced exceedes *log* of the original size of that parse forest. Algorithm 5.5 shows the pseudo-code of the algorithm of this process.

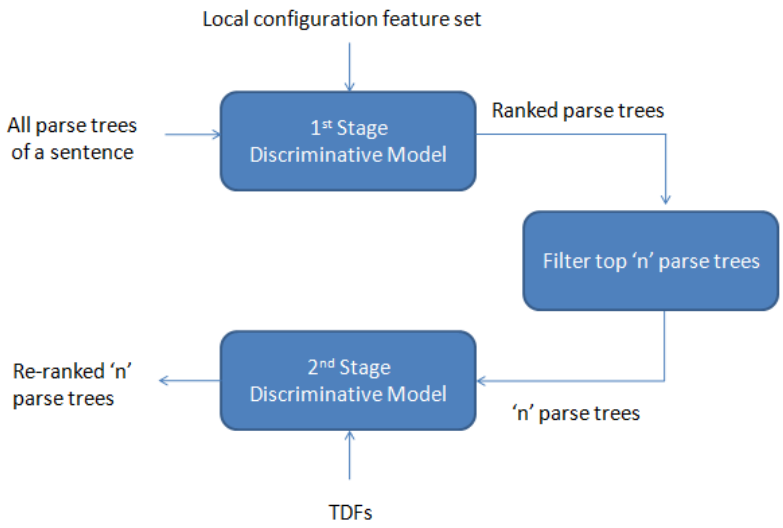


Figure 5.2: Process of re-ranking.

Algorithm 5.5: Parse forest reduction using ranked TDFs

1: $zTDFList$:= List of top $Z\%$ of the ranked TDFs
2: **for** each sentence having $\geq M$ number of syntactic analyses **do**
3: $treeList$:= List of derivation trees of the parse trees of the current sentence
4: $sizeOfParseForest$:= Total number of parse trees of the current sentence
5: **for** each $f \in zTDFList$ where f is the next top-ranked element of the list **do**
6: **if** f is observed in at least L trees of $treeList$ **then**
7: $treeList$:= List of all those trees of $treeList$ where f is observed
8: **end if**
9: **if** number of iteration of the current loop $\geq$ $log_2(sizeOfParseForest)$ OR sizeOf($treeList$) $\leq L$ **then**
10: *exit* from the current loop
11: **end if**
12: **end for**
13: Set $treeList$ as the (reduced) parse forest for the current sentence and continue.
14: **end for**

5.6 *LOGON* System

The *LOGON* infrastructure is a collection of software, grammars, and other linguistic resources (such as *LKB*, *PET*, `[incr tsdb()]`, *ERG*, *TADM*, etc.)[5] to facilitate experimentation with transfer-based machine translation (MT). The central organizing facility for evaluation and testing within the system is `[incr tsdb()]` profiling environment (Oepen, 2001). It provides several specialized tools for regression testing and performance profiling of constraint-based grammars. `[incr tsdb()]` has built-in modules to extract and build discriminative models for the state-of-the-art features (such as local configurations, n-grams, active edges) from the derivation trees. We take advantage of it and do the experiments related to the state-of-the-art features using *LOGON*. The work flow of these experiments is almost same as Figure 5.1 except that there is no use of treebanking decisions.

[5] See http://wiki.delph-in.net/moin/LogonTop for all of these resources.

Chapter 6

Experimentation Environment

In this chapter we will present some of the key details related to the setup of our experiments of what we broadly refer to as our experimentation environment. Roughly speaking, the first few sections of the current chapter will be concerned with aspects of data and features of the experiments. The latter sections will be concerned with aspects of training models and evaluation. We start with the description of both in-domain and out-of-domain data in Section 6.1. Different types of statistics are presented to give a clear picture. Section 6.2 and 6.3 are continuation of more information about treebanking decisions and various feature types from Chapter 4. In these sections, we provide statistical insights about them which are complemented with some additional implementation specific information. Section 6.4 is a continuation of Chapter 2 and 5. While in those chapters we have laid out the mathematical foundations and the detail description about the implementation, this particular section provides information about the specific disambiguation models that are used for the experiments. The chapter concludes in Section 6.5, as we turn to look at some of the various scoring metrics that we incorporate in the system for evaluating performance of the disambiguation models built using different feature sets. These metrics are the key properties that will be used in Chapter 7 to infer the general outcomes of the experiments.

6.1 Data

The data we use for our experiment is a collection of English sentences from the *LOGON parallel tourist corpus* (Oepen et al., 2002) provided by Text Laboratory[1] at the University of Oslo. This data set is also known as *JHPSTG*, an acronym which

[1] See http://www.hf.uio.no/tekstlab/ for more information.

indicates that it is actually built out of three smaller treebanks called *Jotunheimen (JH)*, *Prekestolen (PS)*, and *Turglede (TG)*. The *JH* and *PS* sub-corpora respectively contain approximately *27,000* and *3,700* words in Norwegian, and come with two commissioned translations into English, in addition to the one original English version. The *TG* sub-corpus contains *13,000* words in the Norwegian version, but only comes with two commissioned English translations (i.e. there is no original English version) (Velldal, 2008). *JHPSTG* contains in total *9,410* sentences. We use *8,593* of these sentences for which the parser is able to generate at least one preferred parse tree in the first *500* syntactic analyses. We do not consider any analysis outside of this range in our experiments. *874* of these sentences are held-out as test items and they belong to the *PS* sub-corpus. The remaining *7,719* items of *JH* and *TG* are used for training. The sentences of *JHPSTG* have an average length of *14.68* and average number of *203.26* readings per sentence. We use *ERG* grammar version *LinGO (26-Jan-08)*.

The out-of-domain data are a set of English *Wikipedia* sentences from *WeScience corpus* (Ytrestøl et al., 2009). The corpus comprises a selection of Wikipedia[2] articles in the domain of Natural Language Processing, pre-processed to strip irrelevant markup and segmented into sentence-like units.[3] We use *531* human annotated sentences from the *revision 0.1* of the corpus. The sentences have an average length of *19.38* and average number of *268.69* readings per sentence.

6.2 Decisions taken by the human annotators and system

There are total *30,989* annotated decisions (including duplicates, i.e. multiple occurrences) for all the training items (i.e. *JH* and *TG* sub-corpus) taken by human annotators. The `[incr tsdb()]` system has taken *421,834* inferred decisions (including duplicates) that complement the annotated decisions of the human annotators (refer to Section 4.3 for how these decisions are related). So, the *annotated decisions* comprise only *6.84%* of the total decisions. In fact, many of the *annotated decisions* have also appeared multiple times inside the *inferred decisions*.

We don't create set of distinct/unique decisions to use later for feature extraction. Recall from Section 4.6 that, each set of treebanking decisions per sentence is only used for the parse trees of that particular sentence, which enable the *TDF*s to be highly correlated to the corresponding constructions and corresponding sentence from where they are extracted.

[2] http://www.wikipedia.com
[3] See http://wiki.delph-in.net/moin/WeScience for details.

One important observation for our training data is, almost *99.24%* of the annotated decisions (*30,755* out of *30,989*) are of type *'Yes'*. As we noted in Section 4.4, annotators tend to start with the decisions with the most certainty (Kordoni and Zhang, 2009), so, we can infer from this — annotators prefer more to selecting discriminants than rejecting.

6.3 Feature sets

We build the following feature sets from the training dataset for our experiments —

- **ATDFC**: *TDF*s with context extracted using only *annotated decisions*; total features *27,440*.
- **TDFC**: *TDF*s with context (refer to the Section 4.5 for details); total features *53,362*.
- **YTDFC**: *TDF*s with context extracted using only *'Yes' decisions* (both *annotated* and *inferred*); total features *31,120*.
- **TDF1**: *TDF*s extracted using only template *T1*, in other words, *TDF*s *without* context; total features *29,081*.
- **n-grams**: all *'relevant'* (refer to the Section 4.6) *n-grams* with $n = 4$; total features *438,844*.
- **local configurations**: all *'relevant' local configurations* with *gp = 3 (grandparenting level)*; total features *2,735,486*.
- **active edges**: all *'relevant' active edges*; total features *89,807*.

Notice that, the sizes of all the individual above mentioned *TDF*s feature set are smaller than any of the state-of-the-art feature sets. The value of *'n'* for *n-grams* and *'gp'* of *local configurations* are chosen empirically from the highest accuracy obtained for different values of those parameters in their corresponding feature set's models. Regardless of the types of the features, during the construction of feature set, only unique features are considered. We do not create any feature set for only 'no' decisions as they have few number of occurrences(see the Section 6.2).

6.4 Discriminative models used for the experiments

The main goal of this thesis is to compare various types of features (both the state-of-the-art features and the *TDF*s), and to learn whether treebanking discriminants are potential enough to be used in parse disambiguation model. So, we build separate log-linear training models for *ATDFC*, *YTDFC*, *TDFC*, *TDF1*, local configurations, n-grams and active edges. Recall from Chapter 2 that, log-linear models (Johnson et al., 1999) are considered standard for parse selection for unification-based grammars. Also, recall from Chapter 5 that, event files (which contains features and their occurrence frequency for each reading per sentence) are created before training disambiguation models.

These models serve different purpose for different experiments. Models of *ATDFC* and *TDFC* compare human annotated decisions with all the decisions (i.e. human annotated + system inferred decisions). Models of *YTDFC* and *TDFC* compare preferred decisions with all the decisions (i.e. preferred + non-preferred decisions). Models of *TDF1* and *TDFC* indicate how important the contexts of these decisions are. We previously described the major differences between the *TDF*s and state-of-the-art features. Models of the state-of-the-art features and *TDF*s shed some light on how these differences make distinction among them in terms of efficiency, robustness and informativeness.

We also use model of *TDFC* as re-ranker on top of the ranking done by the model of local configurations. Local configurations allow to obtain highest accuracy among the models of state-of-the-art features (see the Chapter 7). This particular experiment gives us some idea whether model of *TDF*s can perform better as re-ranker.

6.5 Evaluation measures

For each of the discriminative models, we calculate following evaluation metrics from the test data —

- *Exact (match) accuracy*: the *exact match* measure is simply the percentage of times that the top-ranked analysis, by a training model, for each of the sentences in the test data (i.e. the selected analysis) is identical with the reference or gold analysis of the same sentence.

- *5-best (match) accuracy*: this measure is the percentage of times that the five top-ranked analyses for each of the sentences contain the reference or gold analysis.

- *Feature Hit Count (FHC)*: *FHC* is the total number of occurrences of the features (of a particular feature type) inside all the syntactic analyses for all the test sentences. So, for example, if a feature (of a particular feature type) is observed 100 times, then these 100 occurrences are added to total *FHC*.

- *Feature Type Hit Count (FTHC)*: *FTHC* is the total number of *distinct* features (of the corresponding feature type) observed inside the syntactic analyses of all the test sentences.

- *Active Features*: *Active features* are the percentage of the total number of *distinct* features that have been observed inside the test data. In other words, $active\ features = \frac{FTHC}{Total\ number\ of\ features} * 100\ \%$

While *exact* and *5-best* match measures reveal the relative informativeness and robustness of the feature types; *active features*, *FHC* and *FTHC* provide a more comprehensive picture that depicts the relative efficiencies.

Chapter 7

Results and Analyses

Empirical study is indispensable for the evaluation of techniques. To obtain meaningful results, empirical study requires controlled experiments on a considerable amount of data. This chapter presents the results of a series of such experiments using our system which has been introduced in Chapter 5. The collection of experiments as reported here is chosen in order to analyze features extracted using Treebanking Decisions from different points of view, and to illustrate their potentiality to be used for parse disambiguation.

Section 7.1 presents the experiments done to compare performance of disambiguation models of different feature types. Section 7.2 describes scalability of these models on out-of-domain data. Section 7.3 analyzes efficiencies of the feature types. Then, Section 7.4 compares the features extracted using different combination of Treebanking Decisions. Section 7.5 illustrates the experiments of using model of *TDFC* for re-ranking previously ranked parse trees. Finally, Section 7.6 presents the experiments of parse forest reduction using individual top ranked *TDF*s.

Feature templates	**Total features**	**5-best match accuracy**	**Exact match accuracy**
n-gram	438,844	68.19%	41.30%
local configuration	2,735,486	75.51%	50.69%
active edges	89,807	68.99%	41.88%
TDFC	53,362	70.94%	43.59%

Table 7.1: Accuracies obtained on in-domain data using n-grams (n=4), local configurations (with grandparenting level 3), active edges and TDFC.

7.1 Comparison of performances among the models of different feature types

In Section 4.6, we have discussed how *TDF*s differ from the state-of-the-art features. We use disambiguation models of all of these different feature types for the parse selection task to find out whether their differences can make any distinction in their performance. From the experiments on in-domain test data (see Table 7.1), it appears that local configurations achieve highest accuracy among the traditional feature types. But they also use quite a higher number of features (almost *2.7* millions). Note that, features of all the traditional types are tuned using frequency cut-off to obtain a relatively smaller but still reasonable models. For TDFC such tuning is not required (refer to Section 4.6). TDFC does better than both n-grams and active edges, even with a much lower number of features. Though, local configurations gain more accuracy than TDFC, but they do so at a cost of *50* times higher number of features. This indicates that, features extracted using treebanking decisions are more informative. From our analyses on the results, we observe that, TDFC and local configurations perform almost similarly for sentence length $<= 10$, which is quite interesting. This means that, for larger sentences TDFC is missing considerable portion of the structural information of the context which are beyond the words covered by the discriminants of the corresponding decisions (ideally, a discriminant cover only those words which are part of a syntactically ambiguous portion of a sentence). As local configuration features cover all the words (by including all the branches of the derivation trees), they contain all the syntactic labels and grammatical rules applied at a cost of huge number of features.

Feature templates	**5-best match accuracy**	**Exact match accuracy**
n-gram	62.71%	42.37%
local configuration	64.22%	44.44%
active edges	61.77%	39.92%
TDFC	62.71%	41.05%

Table 7.2: Accuracies obtained on out-of-domain data.

7.2 Performance measurement on out-of-domain data

McClosky et al. (2006) noted,

> "Modern statistical parsers require treebanks to train their parameters,

but their performance declines when one parses genres more distant from the training data's domain."

So, we use all the disambiguation models to test on the out-of-domain data (see Table 7.1) to observe the change in their performance. It turns out that, there is a big drop in accuracy for local configurations. Active edges and TDFC also have some accuracy drop. Surprisingly, n-grams do better with the out-of-domain data than in-domain, but still that accuracy is close to that of TDFC. To be precise, n-grams and TDFC achieve equal 5-best match accuracies. It should be noted that, n-grams have *8* times higher number of features than TDFC. Hence, according to these results, we opine that TDFC are more robust, for out-of-domain data, than local configurations and active edges, and almost as good as n-grams.

Feature template	**FHC**	**FTHC**	**Active features**
n-gram	18,245,558	32,425	7.39%
local configuration	62,060,610	357,150	13.06%
active edges	22,902,404	27,540	30.67%
TDFC	21,719,698	17,818	33.39%

Table 7.3: FHC and FTHC calculated for in-domain data

7.3 Active features

While we observe that different feature types have big differences among their number of features, it is predictable that not all of these features are in effect during parse selection. Otherwise, there would have been a big margin among the accuracies obtained by the models. So, we have analysed status of the features of different feature types during parse selection of in-domain data. The outcome of this analysis reveals that, the most important aspect of TDFC is its higher efficiency over its traditional counterparts (Table 7.3). The features of TDFC have much higher number of *active features* than n-grams and local configurations. Recall from Section 6.5, *active features* are the percentage of the total number of *distinct* features that have been observed (as being used during parse selection) inside the test data. These statistics (Table 7.3) indicates that TDFC features have more coverage (as they are comparatively more generic than the traditional features; that is why they have relatively more hits). Despite there is no tuning done on for the features of TDFC using frequency cut-off, unlike traditional features, the treebanking decisions

are themselves restrictive enough to allow to encode comparatively higher active features (refer to Section 4.6).

7.4 Effect of human annotated decisions, contexts and different decision combinations

As mentioned in Chapter 4, we incorporated siblings apart from ancestor information for TDFC. So, we do experiments to find out how much difference contextual information make if they are included in *TDF*s. We also do experiments to evaluate effects of human annotated decisions and positive (i.e. "Yes") decisions. We use features (they will be referred as *TDF1* in the remaining of this document) extracted using only template *T1* (see Section 4.5) to build a disambiguation model and compare its performance with that of TDFC. We also build a separate model using only human annotated decisions (corresponding features will be referred as *ATDFC*), and only positive (i.e. annotated and inferred "Yes" decisions) decisions (corresponding features will be referred as *YTDFC*).

Results indicate that, features extracted from positive decisions have major impact on the accuracy gain. Although, around *60%* of the total features extracted using all annotated and inferred decisions combined are from *"Yes"* decisions, they reach 40.85% exact match accuracy which is near to the accuracy obtained using all decisions. There is a big drop in accuracy (*32.15%*) if contexts are not taken in consideration. Interestingly, though human annotated decisions comprise only *6.84%* of the total decisions, they produce almost *50%* of the total features and their accuracy difference with that of the total decisions is not big.

Feature template	**Total features**	**5-best match accuracy**	**Exact match accuracy**
TDF1	29,081	57.55%	32.15%
TDFC	53,362	70.94%	43.59%
YTDFC	31,120	66.93%	40.85%
ATDFC	27,440	65.22%	38.79%

Table 7.4: Accuracies obtained using different combination of *TDF*. *TDF1* is the *TDF*s extracted using template *T1*; *TDFC* is extracted using the combination of templates *T1*, *T2* and *T3*; *YTDFC* is same as *TDFC* except that the features are collected using only annotated and inferred "Yes" decisions; and finally, *ATDFC* is the collection of *TDF*s extracted using only human annotated decisions.

7.5 *TDF* model as re-ranker

We do experiments to see whether the model of *TDF*s can perform better as a re-ranker. The idea is to do parse disambiguation in two stages. The best model of the state-of-the-art features is used to rank the parse trees of test items in first stage; while in second stage, model of TDFC is used to re-rank top n parse trees per sentence retrieved from the output of first stage ranking. Finally, the best parse tree per sentence is chosen from the output of the second stage re-ranker.

We use the model of local configurations for the first stage as the previous results show local configurations allow to obtain highest accuracy among the models of state-of-the-art features. Recall from Figure 7.1 that, model of local configuration and TDFC achieved accuracy of *50.69%* and *43.59%* respectively and their *5*-best match accuracies are *75.51%* and *70.94%* correspondingly. Figure 7.5 shows the outcome of the re-ranking experiments for top *25*, *10* and *5* parse trees per item obtained from first stage ranker. While the results of re-ranking are better in comparison with that of using TDFC model alone, they could not surpass the accuracies of local configuration model. We observe that, correct parse trees of around *22%* of the test items are missing in the top *25* (when $n = 25$) ranked parse trees by local configuration model; so, they are already out of scope to be considered by the second stage model. For the remaining sentences, we find that, larger sentences are still problematic for the TDFC model despite the parse forest size is considerably reduced by the first stage ranker. This is the primary reason because of which, though there is some improvement in accuracy in comparison the accuracy achieved by stand alone TDFC model, the second stage ranker is not outperforming the local configurations model. Hence, we conclude that, standalone TDFC model might not be useful as re-ranker.

n	**Exact match accuracy**	**5-best match accuracy**
25	43.71%	73.99%
10	44.62%	73.46%
5	45.31%	74.26%

Table 7.5: Accuracies obtained by using the disambiguation model of TDFC as re-ranker.

7.6 Parse forest reduction using individual top ranked features

We sort the *TDF*s according to their weights computed by MaxEnt modelling. Interestingly, when we check the top *20* features to see which feature templates (see Section 4.5) they belong to, we find that all of them are extracted using feature template *T3* (i.e. with contextual information of parent and sibling). We do some experiments to see whether it is possible to reduce the parse forest effectively using top ranked *TDF*s, one at a time, maintaining some constraints (refer to Section 5.5). We vary the percentage of top ranked *TDF*s to be used and the amount of syntactic analyses (i.e. the value of M) that a test sentence must have to be considered for the experiments. As shown in Table 7.6, the results of parse forest reduction are not optimistic. But there are two important observation found in the results – firstly, the features in the top of the rankings become less effective when the depth of these features increased (see the results of 5% and 10% top ranked features); and secondly, parse forests of the sentences with lesser number of syntactic analyses are more likely to be correctly reduced (see the results for M's value 100 and 200). The second finding is consistent with our earlier observations where we mentioned that *TDF*s are more effective to shorter sentences (usually, shorter sentences have less number of syntactic analyses than that of the larger sentences).

% of top ranked features used	***L***	***M***	**Total parse forest reduced**	**Correctly reduced**	**% of the size of forests reduced**
10	25	100	356	72 (20.22%)	8.15%
5	25	100	355	79 (22.25%)	9.25%
10	25	200	309	57 (18.45%)	7.40%
5	25	200	308	64 (20.78%)	8.46%

Table 7.6: Result of parse forest reduction using $Z\%$ of top ranked *TDF*s individually. In the table, by "*% of the size of forests reduced*", we mean the percentage of the original number of total parse trees (of all the reduced parse forests) that remain after reduction.

We will discuss the results of the patterns extraction for correlated discriminants from human decisions in next chapter.

Chapter 8

Extracting Correlated Discriminants from Human Decisions

In the previous chapters, we have focused, more or less, on the potentiality of the discriminative models built using treebanking decisions. While the results of the experiments, presented so far, indicate a good prospect for utilizing treebanking decisions, the types of feature templates that we are using for them are not yet fully conveying cognitive knowledge of the annotators, which we are specifically interested in. This particular chapter is dedicated for this purpose. Our goal is to learn the patterns of decision making process by extracting correlated discriminants from human decisions, if there exists any. Although during annotation, an annotator continues by choosing the discriminants on which he is more confident, there might be a tendency of preferring to choose a certain categories of discriminants (e.g. lexical entry types) before other specific types of discriminants (e.g. head-adjunct schema or head-complement schema). Obviously, such tendency (i.e. choosing specific types of discriminants first) varies from one annotator to the other. `[incr tsdb()]` records the order of discriminant selection of the annotators using timestamp information. However, usually, annotators often make unnecessary choices (i.e. additional discriminants selection) which have no effect on preferred parse tree selection, other than lengthening the decision making process. We assume that, the core decisions (i.e. excluding the additional discriminant selections), required to disambiguate a particular parse forest, have some kind correlation among themselves. So, for example, it may be possible that whenever a human annotator choose a 'hcomp' discriminant to reduce parse forest, it may be most likely that he may also need

to choose a 'subjh' discriminant in one of the next steps to single out the preferred parse tree. The order of discriminant choosing affects the size of the decisions set[1]. If a particular decision is more effective to reduce the parse forest than others, it is likely that if it is chooses prior, then in the later steps fewer decisions will be required to make. For example, choosing 'subjh' right after 'hcomp' may reduce the forest more and require less number of decisions in later steps; while choosing 'subjh' after 2/3 decisions later of 'hcomp' may take more steps. But ultimately, the correlation among these two discriminants will remain the same regardless of their ordering, if both of them are mandatory decisions to be made. We call such correlated discriminants as *pattern* inside human decision making process. Our objective is to extract such patterns. The timestamp information might help us to trace the ordering of the discriminant selection, but the ordering among the correlated discriminant themselves (e.g. 'hcomp' first, then 'hadj' OR 'hadj' first, then 'hcomp') is actually not important for our purpose. So, we have ignored timestamp information for the pattern extraction process. We describe in Section 8.1, how we can extract patterns where we do not consider the ordering of the discriminants inside the corresponding patterns. Following this, we explain how these patterns are used for parse forest reduction in Section 8.2. Finally, we present the results and mention some general observations from these experiments in Section 8.3.

8.1 Pattern extraction

Before we proceed, we would like to define our patterns formally. Let x be such a discriminant that whenever x is observed inside any decision set (which we collect from the training data), that decision set also contains a set of discriminants Z (where $Z \neq \emptyset$), although it might happen that, in some decision set Z is observed despite that decision set does not contain x. We call the set *only* consisting of x and all the elements of Z as *relative unique subset*. We recognize such *relative unique subset*s as *pattern*s, as the occurrence of x is correlated with the elements (i.e. discriminants) of Z. It is implicit in the above definition that, such patterns must contain at least two discriminants.

Here is an example of the *relative unique subset* (i.e. pattern). Let us consider we have the following decision sets –

For the parse forest of sentence 1: {*hadj, hcomp, subjh, hspec*}

For the parse forest of sentence 2: {*hcomp, hadj, subjh, trans_v*}

[1]A decision set for a particular parse forest is the collection of decisions that have been taken to discard all but the preferred parse tree of that forest.

For the parse forest of sentence 3: {*subjh, hadj, hmark*}

For the parse forest of sentence 4: {*hadj, hspec*}

As we can see in the above sets, if a decision set contains *hcomp*, then it also contain {*hadj, subjh*}. Hence, the set {*hcomp, hadj, subjh*} is a *relative unique subset*, i.e. a pattern.

We collect all distict patterns (according to the above definition) from the collection of all decision sets. We obtain total **6151** patterns (i.e. *relative unique subset*) in this way. We call this list of patterns as *coarse grained patterns* (*CGP*).

8.2 Reducing size of the parse forests using patterns

Before using the patterns, we order them based on some criteria. Let *allPatterns*[ ] be the list of patterns, and *pattern1* and *pattern2* are any two elements of the list. Then, *pattern1* will be placed before *pattern2* inside the list if any of the following holds –

(1) total discriminants of *pattern1* > total discriminants of *pattern2*, OR

(2) *pattern1* contains more *HPSG* lexical rules[2] than *pattern2*, OR

(3) *pattern1* contains less negative discriminants (i.e. those discriminants which the human annotators identified as wrong) than *pattern2*.

(1) ensures that we use those patterns first which are more restrictive (the more discriminants inside a pattern, the more constrains to meet for a parse tree to be satisfied as the preferred one). If some discriminants are correlated with a lexical discriminant, then they are likely to be more interrelated (and more specific) than the discriminants of those patterns where all of them are non-lexical discriminant. (2) is used to ensure that we give such patterns more priority. Finally, (3) is used to give priority to the patterns with less negative discriminants.

Once the patterns are sorted, we use them according to their order of appearance (inside the list of all patterns). Algorithm 8.2 shows how reduction of parse forest is done using these patterns.

[2]It should be noted that, all discriminants are basically *HPSG* rules.

Algorithm 8.2: Parse forest reduction by pattern matching

```
1: Let allPatternList[ ] be the sorted list of all patterns
2: Let parseForest[ ] be the collection of all derivation trees of the parse trees of a
   particular sentence
3: Let discriminantSet[ ] be the distinct set of discriminants in all trees of
   parseForest
4: for pattern_i ∈ allPatternList , where 0 ≤ i < sizeOf(allPatternList) do
5:    if pattern_i ⊂ discriminantSet then
6:       tempForest := List of trees of parseForest which contains positive
         discriminants of pattern_i but do not contain the negative discriminants
7:       if tempForest ≠ ∅ then
8:          parseForest := tempForest
9:          if sizeOf(parseForest) = 1 then
10:            return parseForest
11:         end if
12:         Re-populate discriminantSet with the distinct set of discriminants from
            the trees of parseForest
13:      end if
14:   end if
15: end for
16: return parseForest
```

8.3 Experiments and observations

Basically, we construct list of patterns in two ways. The first way is described in Section 8.1. The second way of pattern construction is a little bit different. After collecting the *CGP* from the whole training set, we apply them to each parse forest of training set separately to refine the patterns list further. The idea of refining is — for all the parse forests in training data, if there exists any pattern which reduces[3] the size of a parse forest initially but in doing so discard the preferred parse tree, then this particular pattern should not be part of the pattern list. In other words, we keep only those patterns which do not discard the preferred parse tree of any parse forest when the pattern is applied to the parse forest for the first time at its

[3]If a parse forest becomes empty after using a pattern, then this is not considered as a valid reduction.

initial size (i.e. no reduction is done prior). We call this refined list as *fine grained patterns* (*FGP*). The *FGP* contains total *1614* patterns.

	CGP (1 iteration)	**CGP** (All iterations)	**FGP** (1 iteration)	**FGP** (All iterations)
Total parse forest	786	786	786	786
Reduced parse forest	758	758	26	26
Unaffected parse forests	28	28	760	760
Incorrectly reduced parse forest	410	596	9	9
Correctly reduced parse forest	348	162	17	17
Percentage of correctly reduced parse forest	45.91%	21.37%	65.38%	65.38%
Total parse tress of the reduced parse forest before reduction	156490	156490	8467	8467
Total parse tress of the reduced parse forest after reduction	39248	3399	4034	4034

Table 8.1: Parse forest reduction using *CGP* and *FGP*.

Table 8.1 shows how *CGP* and *FGP* perform when they are used for parse forest reduction. They are applied on *786* items (which have at least two different syntactic analysis) of the test data set. We try to differentiate the effect of reduction after both one iteration (only one pattern is used for reduction) and all iterations (i.e. continue reduction as long as possible according to Algorithm 8.2).

As we can see from the result, *FGP* reduces only limited number of parse forest (26 out of 786). There is no difference between the outputs of one iteration and all iterations. That means, *FGP* is two much restrictive to certain types of construction and there is no mutual relation among the patterns of *FGP* list to do more reduction in those constructions. To put differently, if the pattern set is tuned too finely, there is a high probability that the coverage (i.e. the number of different parse forests to be affected) will be very limited.

For *CGP*, although a large portion of the parse forests are reduced, but most of them are reduced incorrectly. There is a further large drop in correct reduction if more than one pattern are applied on a particular parse forest (though the size of the corresponding forest also shrink considerably). So, it underlines that, it is prudent to reduce forests only once if strong mutual relation between the already applied pattern and next pattern to be applied can not be determined appropriately.

	CGP (1 iteration including negative & lexcial discriminants)	**CGP** (1 iteration without lexcial discriminants)	**CGP** (1 iteration without negative discriminants)	**CGP** (1 iteration without negative & lexcial discriminants)
Total parse forest	786	786	786	786
Reduced parse forest	758	746	690	607
Unaffected parse forests	28	40	96	179
Incorrectly reduced	410	398	409	412
Correctly reduced	348	348	281	195
Percentage of correctly reduced parse forest	45.91%	46.65%	40.72%	32.13%
Total parse tress of the reduced parse forest before reduction	156490	156416	155491	147513
Total parse tress of the reduced parse forest after reduction	39248	40378	37402	27358

Table 8.2: Impact of patterns with negative and lexical discriminants on parse forest reduction.

Table 8.2 shows the impact of lexical discriminants (i.e. *ERG* grammar rules ending with "_le") and negative discriminants (i.e. those discriminants of a pattern which if exist inside a parse tree, the corresponding tree is discarded) on the patterns. As the result indicates, excluding patterns with lexical discriminants do not have any effect on the correctly reduced parse forests, but this ultimately helps to drop the number of incorrectly reduced forests. There is a drastic drop in correct reduction, according to the results, if patterns having negative discriminants are not considered. Recall that, negative decisions comprise a very small portion of the total decisions (see Section 6.2), and yet we see here that they have strong impact.

To conclude, it is obvious from the above results that negative discriminants are quite helpful, though the impact of lexical discriminants is not up to our expectation. The ordering of the patterns is vital (i.e. which pattern should be used prior others). We have tried with parameters such as size of the patterns, existence of the negative discriminants, etc., to sort the pattern list and generalize the ordering for all parse forests. It would be interesting to see if it is possible to make the ordering sensitive

to the construction of the respective sentence (instead of having a fixed order of patterns for all test sentences) whose parse forest is to be reduced.

Chapter 9

Conclusion

9.1 Summary

In this thesis, we examine potentiality of treebanking decisions for the parse disambiguation task. We pose ourselves two questions – *(i)* whether there is any extra (or more) information (in a given treebank) covered by the treebanking decisions which is out of reach of the state-of-the-art feature types, and if so then how that information can be extracted and exploited for addressing the parse disambiguation problem, and *(ii)* whether it is possible to extract hidden correlated patterns of discriminants (if there exists any) from human decisions and to use them for parse forest reduction.

In order to approach the first question, we define three feature templates which captures some key characteristics of treebanking decisions by – relating the lexical words with the discriminants (which are often distant grandparents) without concerning about the intermediate nodes, incorporating siblings of the discriminants which are not recorded in the database but play a vital role for human annotators while making decisions, avoiding exhaustive search (unlike state-of-the-art feature types) by searching only on the sub-trees where the root nodes are discriminants of corresponding decisions, and using both preferred and non-preferred analyses which allows to make use of both positive and negative treebanking decisions. These feature templates do not use treebanking decisions made for the parse forest of one sentence to extract features from the parse forest of another sentence. Consequently, the resulting number of *TDF*s is much smaller than their traditional counterparts. This approach also ensures that *TDF*s remain highly correlated to the corresponding constructions of the corresponding sentences from where they are extracted.

To compare *TDF*s with the traditional feature types. we build their corre-

sponding log-linear models for each of them and test disambiguation performance on both in-domain and out-of-domain data. Results suggest that features extracted using treebanking decisions are more efficient, informative and robust, despite the total number of these features being much less than that of the traditional feature types. We analyze impact of different types of treebanking decisions (yes/no, annotated/inferred). Our findings indicate that, features extracted from positive decisions have major impact on the accuracy gain. We also observe that although human annotated decisions comprise only about one fifteenth of the total decisions, they produce almost half of the total features and their accuracy difference with that of the total decisions is also not so big. Further analyses indicate that, context is an important factor for the *TDF*s, and for larger sentences *TDF*s are missing considerable portion of the structural information of the context that are beyond the words covered by the discriminants of the corresponding decisions.

Some experiments have been done to see whether the model of *TDF*s can perform better as a re-ranker. Results suggest that, as many preferred analyses are already out of consideration due to the failure of the first stage ranker based on traditional features and also as larger sentence are problematic for the *TDF*s, despite the parse forest size is considerably reduced by the first stage ranker, the second stage ranker based on *TDF*s cannot outperform the stand alone best traditional model. Hence, we think disambiguation model built using only *TDF*s might not be useful as a re-ranker.

We also present some experiments which examine whether it is possible to reduce the parse forest effectively using top ranked *TDF*s, one at a time, maintaining some constraints. We vary the percentage of top ranked *TDF*s to be used and the amount of syntactic analyses (i.e. the value of M) that a test sentence must have to be considered for the experiments. We observe that the more is the depth of the list of the top ranked features is used, the less they become effective. Also, results indicate parse forests of the sentences with lesser number of syntactic analyses are more likely to be correctly reduced with top ranked features. The second finding is consistent with our earlier observation that *TDF*s are more effective to shorter sentences (and usually, shorter sentences have less number of syntactic analyses than that of the larger sentences).

To approach the second question, we extract *relative unique subset*s from the training dataset. We consider such subsets as *coarse grained pattern*s and generate a more refined list of *fine grained patterns*. We sort both the lists of patterns based on some criteria. Experimental results indicate that, if the pattern set is tuned too finely, there is a high probability that the coverage will be very limited. The results

suggest that, correct reduction accuracy drops if multiple number of patterns are applied on a particular parse forest, though the size of the corresponding forest also shrink considerably. It underlines that, it is prudent to reduce forests only once if strong mutual relation between the already applied pattern and next pattern to be applied can not be determined appropriately.

To summarize, the different types of experiments and their outcomes suggest that, treebanking decisions are indeed capable of contributing in parse disambiguation. However, as the number of treebanking decisions are usually not so big (which limits the number of features to be generated using them) and also they are subject to only some specific parts of the sentence constructions, disambiguation models built using them alone often cannot distinguish analyses of the large sentences properly.

9.2 Comparison to the related works

To our best knowledge, there has been no work reported until now on using the treebanking decisions for the task of parse disambiguation. Hence, we are unable to compare our study directly to other works. Nevertheless, we compare our extract treebanking decisions features with the state-o-the-art feature types that have been used by other researchers (Toutanova et al., 2002; Baldridge and Osborne, 2003; Osborne and Baldridge, 2004; Toutanova et al., 2005; Crysmann, 2008). We use the same settings of log-linear models that they use. The data that we use are also from the same Redwoods-style treebanks. We demonstrate that our features are are more efficient, informative and robust. Those previous researches did experiments using combined or ensembled-based models which we have not done in this thesis and leave as a future work.

These previous studies have reported relatively high exact match accuracies with earlier versions of *ERG* (Flickinger, 2000) on datasets with very short sentences. With much higher structural ambiguities in *LOGON parallel tourist corpus* (Oepen et al., 2002) and *WeScience corpus* (Ytrestøl et al., 2009) sentences and the more enriched recent version of *ERG*, the overall disambiguation accuracy will drop significantly.

To be more precise, Toutanova et. al. (2002) did experiments over 5312 sentences (having average length 7.0 and structural ambiguity 8.3) of the 1^{st} growth of the Redwoods treebank. They obtained as much as 82.65% exact match accuracy for parse selection using a combined model of several discriminative models. They did similar type of experiments over 5266 sentences of 3^{rd} growth of the Redwoods.

The highest accuracy they could reach was 76.7% (Toutanova et al., 2005). So, there is a big drop in accuracy. This is due to increased ambiguity of those 3^{rd} growth sentences which have average length of 9.1 and structural ambiguity of 57.8. As mentioned in Chapter 3, Osborne and Baldridge (2004) adopted ensemble based active learning for parse selection. They experimented on 5302 sentences (with avg. length 9.3 and avg. no. of parses 58.0) of 3^{rd} growth of Redwood treebank, and reported 77% exact match accuracy. All of these works used *ERG* but much earlier version. We use *ERG* grammar version *LinGO (26-Jan-08)*. Our in-domain sentences have an average length of *14.68* and average number of *203.26* readings per sentence (and the out-of-domain sentences have an average length of *19.38* and average number of *268.69* readings per sentence). Using *only* the treebanking decision features, we obtain 43.59% accuracy (refer to Section 7.1) for the in-domain data.

Related works on parser adaptation and re-ranking, briefly introduced in Section 3.3, do not consider cognitive aspects of the fine-grained decision-making process of the human annotators. So, we cannot compare with them, too. We show that decisions taken during treebank annotation can produce good candidate features to be used for parser adaptation, though they might be needed to be combined with other traditional features.

9.3 Open questions and future works

The results of the experiments described in this paper indicate a very good prospect for utilizing treebanking decisions, although, we think that the types of feature templates that we are using for them are not yet fully conveying cognitive knowledge of the annotators, in which we are specifically interested in. For instance, we expect to model human disambiguation process more accurately by focusing only on human annotators' decisions (excluding inferred decisions). Such a model will not only improve the performance of the parsing system at hand, but can also be applied interactively in treebanking projects to achieve better annotation speed (e.g., by ranking the promising discriminants higher to help annotators make correct decisions).

We have several open questions after this study. Firstly, *what types of other decisions/consideration does an annotator make which is not recorded inside the log files but can be recovered using the accepted and discarded discriminants through introspection?*[1]

Secondly, *how can we include the coverage of the parts of sentence constructions*

[1] An example of such considerations is the siblings of discriminants which we already use in this thesis.

that are not covered by the discriminants? To put differently, we want to know whether there is any way to increase the locality of the features. Surely, increasing the locality will boost disambiguation accuracy, at least for the larger sentences.

The third question is, as we have discussed in Chapter 8, *is it possible to make the ordering of the patterns sensitive to the construction of the respective sentence (instead of having a fixed order of patterns for all test sentences) whose parse forest is to be reduced?* A related question would be, whether we could refine the list of *coarse grained patterns* which would produce a list of patterns somewhere in the middle (i.e. not too finely tuned) between the list of *coarse grained patterns* and the list of *fine grained patterns*.

The fourth question is, *does there exist any better modelling technique, which resembles the process of treebanking, other than maximum entropy modelling?* From a high level point of view, the working process of Redwoods-style treebanking is almost similar as Decision Tree (DT) modelling. In DT modelling, each internal node represents a particular attribute. For different possible values, each of these attributes split the event space[2] into separate disjoint branches (Mitchell, 1997). Attributes in different levels of the sub-trees of DT are selected based on the information gain[3] computed, for the remaining attributes, over the events which are covered by the corresponding sub-tree. The terminal nodes of DT are labeled with various possible outcomes; in case of binary DT (i.e. each feature with maximum two possible values) for binary classification, some nodes are classified as 'Yes'/'Accepted' and the others 'No'/'Rejected'. In Redwoods-style treebanking, the treebanking decisions resembles the attributes of the DT. For choosing the appropriate discriminant (which is later saved as annotated decision), human annotators rely on their cognitive rather than calculating information gain. But there are two major differences between these two approaches, which differentiate them from each other, yet indicates the opportunity where one can take the advantages of the other. The signature characteristic of Redwoods-style treebanking is, it is a greedy search. Each time a decision is made, only those parse trees are kept for consideration (for the next decision) which are accepted by that decision. Because, the assumption is – only one (in some special cases multiple) of the parse trees is correct. But the limitation of the human annotators is, it is not guaranteed that the decision taken in a particular step is the optimal decision, i.e. which rejects the maximum number of non-preferred parse trees. Hence, the total number of decisions could be more than

[2]In case of parse selection, the events represent different syntactic realization of a particular sentence.

[3]The expected value of the information gain is the mutual information $I(X;A)$ of X and A – i.e. the reduction in the entropy of X achieved by learning the state of the random variable A.

the size of the optimal decision set, and the depth of the tree could be more than optimal depth. This is where DT has an upper hand. The information gain calculated for the features at a particular step helps to choose the best attribute of that step which ultimately limits the whole DT in minimal length. We think, this is a potential area of future research. If it is possible for the system to take (or at least suggest) optimal decisions (using information gain or some other measures) during treebank annotation, the whole annotation process will be faster and efficient.

The final and perhaps the most important question is, *how will the treebanking decisions features perform if they are combined with the traditional features to build a single disambiguation model?*[4] But we believe such model will perform better than the existing models. Because, A treebanking decisions feature represents partial information about the right parse tree (as most usual features). But in some way, it also indicates that it is a point of a decision (point of ambiguity with respect to the underlying pre-processing grammar), hence carrying some extra bit of information.

To conclude, this thesis has introduced the novel notion of utilizing cognitive aspects of treebanking. The positive outcome of this thesis regarding using treebanking discriminants as parse disambiguation features leaves many places open to extend, improve or refine various parts of the specific instantiation adopted in this work. With this last statement, we would like to end up our exploration for this thesis work.

[4]We have left it as a future work rather than doing in this thesis as this would require to convert the huge amount of Lisp codes of the LOGON system to Java before integrating in our system.

Appendix A

Usage of the System developed in this Thesis

Download link

The implementation of the thesis can be downloaded from *http://fmchowdhury.googlepages.com/downloads* under GNU General Public License.

Prerequisites

To run the system, the following prerequisites must be installed beforehand —

- *Java Runtime Environment* (JRE) 1.6 or higher, available in *http://www.java.com/en/*.
- *The Toolkit for Advanced Discriminative Modeling* (TADM), available in *http://tadm.sourceforge.net/*.

Installation steps

The following steps are independent of the operating system types. We assume the user is using command prompt or shell or terminal. We further assume the user is familiar with the basic commands of the command prompt or shell that are necessary to complete the steps below.

- Retrieve and save the lexicon for the desired version of the HPSG grammar that has been used for the annotation of the Redwoods-style treebank data that is going to be used for the experiments. The file must be in written in according to the specification of the *Type Description Language* (TDL; (Krieger and Schäfer, 1994)). For ERG grammar, the equivalent file *lexicon.tdl* is available inside the donwloadable zipped file in *http://www.delph-in.net/erg/*.
- Download the *Tedot.tar.gz* file from the above mentioned link.

- Unzip the file.
- Access into the directory *Tedot*
- Open the file *Settings.txt*
- Set the full path of the training profile according to the example given inside *Settings.txt*.
- Set the full path of the test profile.
- Set the full path of the lexicon file.
- Set the full path of the TADM installation directory.
- Save the file *Settings.txt* and close it.
- Run the command *javac TedotMain.java* (assuming JRE is already installed and JAVAPATH is set). This will create all the compiled .class files.
- To run the system, just run the command *java -Xms32m -Xmx2000m TedotMain*.

How to use the system

It should be noted that, this implementation only considers the Redwoods-style treebank profiles. Both training and test profiles (and sub-profiles) must have at least the following files — *parse, result, decision, item, relations* and *preference*. The usage of the system is fairly easy. After running the system, it will show various options, e.g. "3. Extract features" or "9. Count feature hit (FHC) and feature type hit (FTHC)". Each option is followed by the message about the basic purpose of that particular option. Type the number (in case of the previous examples – 3 or 9) on the left side of the options and press Enter button. There will be a full set of instructions on the screen how to proceed further. All the results and analyses are written inside files. The system will automatically ask for the name of the file to write from the user. There will be also some general comments about how to interpret the results written inside the file.

How to enhance/modify the Source Code

All the classed, methods and variables inside the source files (i.e. *.java* files) are documented. However, for any kind of further clarification, the user is encouraged to send enquiries in *fmchowdhury@gmail.com*.

References

Steven Abney. 1997. Stochastic attribute-value grammars. *Computational Linguistics*, 23:597–618.

Hiyan Alshawi. 1990. Resolving quasi logical forms. *Computational Linguistics*, 16(3):133–144.

Jason Baldridge and Miles Osborne. 2003. Active learning for HPSG parse selection. In *Proceedings of the 7th Conference on Natural language learning at HLT-NAACL 2003*, pages 17–24.

Adam L. Berger, Vincent J. Della Pietra, and Stephen A. Della Pietra. 1996. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71.

Ulrich Callmeier. 2001. Efficient parsing with large-scale unification grammars. Master's thesis, Universität des Saarlandes, Saarbrücken, Germany.

David Carter. 1997. The TreeBanker: A tool for supervised training of parsed corpora. In *Proceedings of the Workshop on Computational Environments for Grammar Development and Linguistic Engineering*, Madrid, Spain.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL 2005)*, Michigan, USA.

Eugene Charniak. 2000. A maximum entropy-based parser. In *Proceedings of the 1st Annual Meeting of the North American Chapter of Association for Computational Linguistics (NAACL 2000)*, pages 132–139, Seattle, USA.

Stanley F. Chen and Ronald Rosenfeld. 1999. A gaussian prior for smoothing maximum entropy models. Technical Report Technical Report Number CMUCS-CS-99-108, Carnegie Mellon University.

Ann Copestake, Dan Flickinger, Carl Pollard, and Ivan A. Sag. 2006. Minimal Recursion Semantics: an introduction. *Research on Language and Computation*, 3(4):281–332.

Ann Copestake. 2002. Implementing typed feature structure grammars. In *CSLI Publications*, Stanford, CA, USA.

Thomas M. Cover and Joy A. Thomas. 1991. *Elements of information theory*. Wiley-Interscience, NY, USA.

Berthold Crysmann. 2008. Parse selection with a German HPSG grammar. In *Proceedings of the Workshop on Parsing German*, pages 9–15, Columbus, Ohio, June. Association for Computational Linguistics.

Murat Ersan and Eugene Charniak. 1995. A statistical syntactic disambiguation program and what it learns. *Learning for Natural Language Processing*, 1040:146–159.

Dan Flickinger. 2000. On building a more efficient grammar by exploiting types. *Natural Language Engineering*, 6 (1) (Special Issue on Efficient Processing with HPSG):15–28.

Stuart Geman and Mark Johnson. 2002. Dynamic programming for parsing and estimation of stochastic unification-based grammars. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL 2002)*, pages 279–286, Philadelphia, USA.

Yusuke Miyao Hara, Tadayoshi and Jun'ichi Tsujii. 2007. Evaluating impact of re-training a lexical disambiguation model on domain adaptation of an HPSG parser. In *Proceedings of IWPT 2007*, Prague, Czech Republic, June.

Tadayoshi Hara, Yusuke Miyao, and Jun'ichi Tsujii. 2005. Adapting a probabilistic disambiguation model of an HPSG parser to a new domain. In *Proceedings of the 2nd International Joint Conference on Natural Language Processing (IJCNLP 2005)*, pages 199–210, Jeju Island, Korea, October.

Mark Johnson, Stuart Geman, Stephen Canon, Zhiyi Chi, and Stefan Riezler. 1999. Estimators for stochastic unifcation-based grammars. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL 1999)*, pages 535–541, Maryland, USA.

Jin-Dong Kim, Tomoko Ohta, Yuka Tateisi, and Jun'ichi Tsujii. 2003. GENIA corpus - a semantically annotated corpus for bio-textmining. *Bioinformatics*, 19(1):180–182.

Valia Kordoni and Yi Zhang. 2009. Annotating wall street journal texts using a hand-crafted deep linguistic grammar. In *Proceedings of the 3rd Linguistic Annotation Workshop (LAW III)*, Singapore.

Hans-Ulrich Krieger and Ulrich Schäfer. 1994. TDL: A type description language for constraint-based grammars. In *Proceedings of the 15th conference on Computational linguistics (COLING 1994)*, pages 893–899.

Robert Malouf and Gertjan Van Noord. 2004. Wide coverage parsing with stochastic attribute value grammars. In *Proceedings of the 1st International Joint Conference on Natural Language Processing (IJCNLP 2004) Workshop: Beyond shallow analyses – formalisms and statistical modeling for deep analyses*, Sanya City, Hainan Island, China.

Robert Malouf. 2002. A comparison of algorithms for maximum entropy parameter estimation. In *Proceedings of the 6th Workshop on Computational Language Learning (CoNLL 2002)*, Taipei, Taiwan. Association for Computational Linguistics.

David McClosky and Eugene Charniak. 2008. Self-training for biomedical parsing. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies (HLT-ACL 2008)*, pages 101–104.

David McClosky, Eugene Charniak, and Mark Johnson. 2006. Reranking and self-training for parser adaptation. In *Proceedings of the 21st International Conference on Computational Linguistics*, pages 337–344, Sydney, Australia.

Osborne Miles. 2000. Estimation of stochastic attribute-value grammars using an informative sample. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING 2000)*, pages 586–592, Saarbrücken, Germany.

Thomas M. Mitchell. 1997. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA.

Yusuke Miyao and Jun'ichi Tsujii. 2002. Maximum entropy estimation for feature forests. In *Proceedings of the 2nd International Conference on Human Language Technology Research*, pages 292–297, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Yusuke Miyao and Jun'ichi Tsujii. 2005. Probabilistic disambiguation models for wide-coverage HPSG parsing. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics (ACL 2005)*, pages 83–90.

Stefan Müller and Walter Kasper. 2000. HPSG analysis of German. In Wolfgang Wahlster, editor, *Verbmobil: Foundations of Speech-to-Speech Translation*, pages 238–253. Springer, Berlin.

Joakim Nivre, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 2007. The CoNLL 2007 shared task on Dependency parsing. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 915–932, Prague, Czech Republic, June 28-30.

Stephan Oepen and Dan Flickinger. 1998. Towards systematic grammar profiling test suite technology ten years after. *Computer Speech and Language*, 12(4):411–436.

Stephan Oepen, Kristina Toutanova, Stuart Shieber, Christopher Manning, Dan Flickinger, and Thorsten Brants. 2002. The LinGO Redwoods treebank: motivation and preliminary applications. In *Proceedings of the 19th International Conference on Computational Linguistics (COLING 2002)*, pages 1–5, Taipei, Taiwan.

Stephan Oepen, Helge Dyvik, Jan Tore Lønning, Erik Velldal, Dorothee Beermann, John Carroll, Dan Flickinger, Lars Hellan, Janne Bondi Johannessen, Paul Meurer, Torbjørn Nordgård, and Victoria Rosén. 2004. Som å kapp-ete med trollet? Towards MRS-based Norwegian-English machine translation. In *Proceedings of the 10th International Conference on Theoretical and Methodological Issues in Machine Translation*, pages 11–20, Baltimore, MD, USA.

Stephan Oepen. 2001. [incr tsdb()] — competence and performance laboratory. User manual. Technical report, Computational Linguistics, Saarland University, Saarbrücken, Germany.

Miles Osborne and Jason Baldridge. 2004. Ensemble-based active learning for parse selection. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL 2004): Main Proceedings*, pages 89–96, Boston, Massachusetts, USA.

Adwait Ratnaparkhi. 1997. A simple introduction to maximum entropy models for natural language processing. Technical report, Institute for Research in Cognitive Science, University of Pennsylvania.

Adwait Ratnaparkhi. 1998. *Maximum entropy models for natural language ambiguity resolution*. Ph.D. thesis, University of Pennsylvania, PA, USA.

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 Annual Meeting of the Association for Computational Linguistics (ACL 2002)*, pages 279–286, Philadelphia, USA.

Claude E. Shannon. 1948. A mathematical theory of communication. *The Bell System Technical Journal, 27*, pages (pt.1) and 623 – 656 (pt.2).

Yuka Tateisi, Akane Yakushiji, Tomoko Ohta, and Jun'ichi Tsujii. 2005. Syntax annotation for the GENIA corpus. In *Proceedings of the 2nd International Joint Conference on Natural Language Processing (IJCNLP 2005)*, pages 222–227, Jeju Island, Korea, October.

Kristina Toutanova, Christoper D. Manning, Stuart M. Shieber, Dan Flickinger, and Stephan Oepen. 2002. Parse ranking for a rich HPSG grammar. In *Proceedings of the 1st Workshop on Treebanks and Linguistic Theories (TLT 2002)*, pages 253–263, Sozopol, Bulgaria.

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 2003)*, Cavtat-Dubrovnik, Croatia, September.

Kristina Toutanova, Christoper D. Manning, Dan Flickinger, and Stephan Oepen. 2005. Stochastic HPSG parse selection using the Redwoods corpus. *Journal of Research on Language and Computation*, 3(1):83–105.

Erik Velldal. 2008. *Empirical Realization Ranking*. Ph.D. thesis, University of Oslo, Oslo, Norway.

Gisle Ytrestøl, Stephan Oepen, and Dan Flickinger. 2009. Extracting and annotating wikipedia sub-domains. In *Proceedings of the 7th International Workshop on Treebanks and Linguistic Theories*, pages 185–197, Groningen, the Netherlands.

Yi Zhang, Valia Kordoni, and Erin Fitzgerald. 2007. Partial parse selection for robust deep processing. In *Proceedings of the ACL 2007 Workshop on Deep Linguistic Processing*, pages 128–135, Prague, Czech Republic, June.