Robust Deep Linguistic Processing

Dissertation
zu Erlangung des akademischen Grades eines
Doktors des Philosophie
der Philosophischen Fakultäten
der Universität des Saarlandes

vorgelegt von

Yi Zhang

aus Shanghai, China

Saarbrücken, 2008

Die Dekanin:	Prof. Dr. Susanne Kleinert
Berichterstatter/innen:	Prof. Dr. Hans Uszkoreit
	PD Dr. Evagelia Kordoni
Tag der letzten Prüfungsleistung:	12.12.2007

Abstract

This dissertation deals with the robustness problem of deep linguistic processing. Hand-crafted deep linguistic grammars provide precise modeling of human languages, but are deficient in their capability of handling ill-formed or extra-grammatical inputs. In this dissertation, we argue that with a series of robust processing techniques, improved coverage can be achieved without sacrificing efficiency or specificity of deep linguistic processing.

An overview of the robustness problem in state-of-the-art deep linguistic processing systems reveals that insufficient lexicon and over-restricted constructions are the major sources for the lack of robustness. Targeting both, several robust processing techniques are proposed as add-on modules to the existing deep processing systems.

For the lexicon, we propose a deep lexical acquisition model to achieve automatic online detection and acquisition of missing lexical entries. The model is further extended for acquiring multiword expressions which are syntactically and/or semantically idiosyncratic. The evaluation shows that our lexical acquisition results significantly improved grammar coverage without noticeable degradation in accuracy.

For the constructions, we propose the partial parsing strategy to maximally recover the intermediate results when the full analysis is not available. Partial parse selection models are proposed and evaluated. Experiment results show that the fragment semantic outputs recovered from the partial parses are of good quality and high value for practical usage. Also, the efficiency issues are carefully addressed with new extensions to the existing efficient processing algorithms.

Zusammenfassung

Diese Dissertation befasst sich mit dem Robustheitsproblem tiefer Sprachverarbeitungssysteme. Manuell erstellte tiefe Grammatiken liefern eine präzise Modellierung menschlicher Sprache, sind aber unzureichend hinsichtlich ihrer Möglichkeiten, falsch aufgebaute oder zusätzliche grammatische Eingaben zu verarbeiten. In dieser Dissertation werden wir zeigen, dass eine verbesserte Abdeckung mit einer Reihe von robusten Verarbeitungstechniken erreicht werden kann, ohne dabei die Effizienz oder auch die Exaktheit tiefer Sprachverarbeitung zu opfern.

Ein Überblick über das Robustheitsproblem bei aktuellen Sprachverarbeitungssystemen zeigt, dass sowohl ein unzureichendes Lexikon als auch zu restriktive Konstruktionen die Hauptursachen für mangelnde Robustheit darstellen. Es werden einige robuste Verarbeitungstechniken als Zusatzmodule für die bestehenden tiefen Verarbeitungssysteme vorgeschlagen, die bei diese beiden Ursachen ansetzen.

Hinsichtlich des Lexikons schlagen wir ein tiefes lexikalisches Akquisitionsmodell vor, um eine automatische Onlineerkennung und -akquisition fehlender lexikalischer Einträge zu erreichen. Außerdem wird das Modell um eine Akquisitionsfunktion für Multiwortausdrücke erweitert, die syntaktisch und/oder semantisch idiosynkratisch sind. Die Auswertung zeigt, dass unsere lexikalische Akquisition eine wesentlich verbesserte Grammatikabdeckung ohne erkennbaren Genauigkeitsverlust erreicht.

Für die Konstruktionen schlagen wir die partielle Parsingstrategie vor, um die Zwischenergebnisse möglichst umfassend wiederherzustellen, wenn eine vollständige Analyse nicht verfügbar ist. Es werden partielle Parse-Selektionsmodelle vorgestellt und bewertet. Versuchsergebnisse zeigen, dass die fragmentierten semantischen Ausgaben,

die über die partiellen Parse wiederhergestellt worden sind, eine hohe Qualität und einen hohen Gebrauchswert aufweisen. Auch werden Effizienzfragen detailliert mittels neuer Erweiterungen zu den bestehenden effizienten Verarbeitungssystemen betrachtet.

Acknowledgments

I am deeply grateful to my supervisors Hans Uszkoreit and Valia Kordoni for their continuous support and advise regarding the work reported in this dissertation. I also thank them for providing a wonderful working environment, where I had the chance to meet many others who have contributed to this work. Among them, my sincere thanks go to Dan Flickinger for his continuous support on the grammar, prompt feedback, and inspiring discussions. I thank Timothy Baldwin, John Carroll, Stephan Oepen, Aline Villavicencio, for sharing ideas, helping with experiments, and providing healthy criticisms. My thanks go to Francis Bond, Berthold Crysmann, Bernd Kiefer, Ulrich Schäfer, Melanie Siegel, Erik Velldal, Benjamin Waldron, Feiyu Xu, for their support on various grammar resources and processing software. My thanks also go to Valia Kordoni, Rebecca Dridan, Jeremy Nicholson, who have helped to proofread the dissertation and generously offered numerous corrections and suggestions. Of course, any remaining errors are mine. I am grateful to the DFG scholarship in the International Graduate College "Language Technology and Cognitive Systems", which has provided generous financial support during my study. A final word of gratitude is for my mother, Heping Li, and my wife, Yu Chen, for their unwavering support, without which I would not have been willing or able to find my place in this profession.

Contents

List of Figures

List of Tables

1 Introduction

I show you doubt, to prove that faith exists.

— Robert Browning *(1812 - 1889)*

The ideas in this dissertation grew out of my experience with grammar development and my attempts at building applications based on such grammars. My first experience with deep linguistic grammars was during a short visit to Saarbrücken before I started my PhD studies. Back then, I was not familiar with large scale linguistically motivated grammars, and was instantly fascinated by how linguistic studies can be formally described and implemented, and the potential applications of such promising language resources. Soon I decided to move to Saarbrücken to pursue a PhD degree working on deep linguistic processing. I started by taking on an ambitious attempt to implement a `HPSG` grammar for Mandarin Chinese using the `DELPH-IN` resources. This was partly because there was no large scale `HPSG` grammar for Chinese at that time, but another more practical reason was to familiarize myself with the deep processing tools. After struggling through months of frustration, I managed to construct a sketch of a small grammar that covers the basic constructions. By then I realized that writing a grammar of reasonable size would probably take years, if not decades, particularly because there is a lack of systematic theoretical study of `HPSG` for Chinese specific language phenomena. In a retrospect at this point, I found myself to be nowhere near my initial intention of building a language resource that can be useful for applications. The doubt led to a quick retreat. Losing certainty about deep linguistic processing in general, I looked for comfort from existing large grammars. It did not take me long to realize that even with the largest grammars that represent the state of the art grammar engineering results, various problems exist when one tries to use them in real applications. It is not just a coincidence that deep linguistic processing has been a disfavored approach for a long time. The most prominent problem among others is a lack of robustness. It occurred to me that it would be a more interesting topic for me to search for solutions to this problem. Following this thread, I have been working on robust deep processing techniques since that time, and most of the work made its way into this dissertation.

This dissertation describes a series of techniques that lead towards robust deep linguistic processing. In this chapter, I will define the concept of deep linguistic processing, followed by an overview of the state of the art deep linguistic processing platforms, as well as the main challenges it is faced with. Finally, the structure of the dissertation is outlined at the end of the chapter.

1.1 Deep Linguistic Processing

Deep linguistic processing (*DLP*), or *deep processing*, is concerned with natural language processing approaches that aim at modeling the complexity of natural languages in rich linguistic representations. Such approaches are related to either a particular linguistic theory (e.g., `CCG`, `HPSG`, `LFG`, `TAG`, Prague School), or a more general framework which draws on theoretical and descriptive linguistics. Traditionally, deep linguistic processing has been concerned with grammar development for parsing and generation, with many deep processing systems using the same grammar for both directions. Being grammar centric, the studies of deep linguistic processing mainly focus on two questions:

- How to develop linguistically motivated deep grammars?
- How to effectively utilize the knowledge in the given deep grammars to achieve the application tasks?

The first question leads to a whole sub-field of study in grammar engineering, while the second question is closely related to processing techniques not limited to the deep processing community (i.e., parsing, generation, etc.).

1.1.1 Grammar

Grammar is the study of rules governing the use of language. Systematic studies of grammars started thousands of years ago and the methodology has been constantly evolving over the time. Since the 1950s, a new branch of language study named computational linguistics, has emerged as a new field and opened up several novel ways of grammar study. Among them, the approach to describe natural language with formal grammars has been widely attended, with both fruitful success and miserable setbacks.

Formal grammar is an abstract structure that describes a formal language precisely. Though doubts as to whether formal grammar is capable of describing human languages have always been around, it has never impeded the ambitious attempts of building large-scale formal grammars for various human languages. Some of the earlier approaches managed to achieve reasonable coverage and/or accuracy on sub-languages for specific applications. More recent approaches aim at both broad coverage and high accuracy of languages without domain constraints, for both parsing and generation tasks.

While the development of grammars were taking place, researchers soon realized that the growth of the grammars heavily depends on the description language of the grammar, the formalism framework. The quest for a better, more powerful, while computationally affordable framework soon branched into various grammar formalisms. The choice of different grammar formalisms leads to the later blossom of various linguistic theories: transformational-generative grammar (TGG), categorial grammar (CG), dependency grammar (DG), tree-adjoining grammar (TAG), lexical functional grammar (LFG), generalized phrase structure grammar (GPSG), head-driven phrase structure grammar (HPSG), just to name a few of them.

Despite the differences among different frameworks, grammar development is almost always a painstaking task. Especially when aiming for both broad-coverage and high precision, it usually takes years (if not decades) before the grammar can reach a reasonable size. Also, due to the strong cohesion in language phenomena, a slight change of the grammar in one aspect might result in dramatic changes in other corners. This makes it hard to modularize the task of grammar development. Large grammars are typically written by very few linguists continuously over decades. Distributed parallel grammar development is very difficult in practice, if possible, at all. Nevertheless, the continuous work on grammar engineering has seen fruitful outcomes in recent years. Details about the latest achievements in grammar engineering will be discussed in Section 1.2.

It is worth noting that another relatively new approach of grammar development has emerged in recent years. Instead of hand-crafting the grammar, the approach extracts or induces the grammar from annotated corpora (i.e., treebanks) with much less human intervention. In such an approach, the main effort shifts to the creation of large amounts of annotated data. This is achieved by either setting up a good annotation guideline and employing multiple human annotators, or semi-automatically converting the existing treebanks into annotations that are compatible with the underlying grammar framework and include richer linguistic information. The grammars created by such methods usually take shorter development time and the performance of the grammar can be fine-tuned by either expanding the treebank or by improving the extraction algorithm. The main problem with this approach is that the grammars are usually less accurate from two aspects.

First, the "depth" of the grammar is largely dependent on the annotation. Asking human annotators to label detailed linguistic information on the treebank is very difficult, and will inevitably lead to low inter-annotator agreement. The semi-automatic conversion approach requires the existence of multiple linguistic resources, and their inter-compatibility.

Second, the treebank induced grammars usually overgenerate massively. It is typically the case that only grammatically well-formed sentences are annotated in the treebank. Therefore, the induced grammar does not have a strict sense of grammaticality. And the resulting grammar produces a huge amount of analyses per input, not all of which are correct. For parsing tasks, the correct analysis is selected by a parse disambiguation model. But such grammars are less suitable for generation tasks.

In this dissertation, we focus on the deep linguistic processing techniques that rely on hand-crafted deep grammars, simply because they are such distinct grammar resources that provide accurate modeling of human languages.

1.1.2 Processing

Given a grammar, either hand-crafted or treebank induced, it requires extra processing techniques to utilize the encoded linguistic knowledge. Typically, there are two types of tasks in which the grammar is used: parsing and generation.

The parsing task is concerned with converting natural language strings to linguistically annotated outputs. In deep linguistic parsing, the output contains not only basic syntactic information, but often semantic analysis, as well. The exact output annotation varies a lot depending on the framework, but they normally share the properties of exploring a huge solution space. This requires various efficient processing techniques to facilitate the search for (either exact or approximate) best results.

In the generation task, the processing goes in the opposite direction. The output of the processing is the natural language utterances, while the input is the abstract (semantic) representation of the meaning. Similar efficiency and specificity challenges exist for generation tasks, but now the disambiguation model needs to select the best natural language utterances.

It should be noted that the processing techniques in use for a specific task are largely dependent on the characteristics of the grammar. For grammars aiming at high precision, the coverage is usually low, hence robust processing techniques are necessary. For grammars aiming at broad-coverage, overgeneration is often a problem, therefore a more sophisticated disambiguation step is of higher importance. For grammars that aim at both, a mixture of different techniques is needed to achieve a balanced performance. It should also be noted that, even with the same grammar, when used in different application tasks, different configurations of the processing modules should be used to achieve optimal functionality.

1.1.3 “Deep” vs. “Shallow”

The term *“deep linguistic processing”* intends to differentiate the strongly linguistic theory-driven processing techniques we have discussed from those approaches which are less linguistically driven. The latter class of approaches are referred to as “shallow” processing techniques, for they usually concentrate on specific language phenomena or application tasks without thoroughly modeling of the language. By this definition, processing techniques like part of speech tagging, named entity recognition, and phrase chunking all belong to “shallow” processing.

However, it should be pointed out that there is no absolute boundary between deep and shallow processing techniques. Rather, the terms “deep” and “shallow” should be taken in a relative sense. Even within the well-acknowledged deep processing communities, some frameworks provide more detailed analyses than others.

Also, the shallow processing techniques do not need to be separated from deep linguistic processing. In many cases, they can complement each other and combine together to achieve better application performance. Such combination is called *hybrid natural language processing* (Uszkoreit, 2002; Callmeier et al., 2004). But this is beyond the focus of this dissertation.

1.2 State of the Art Deep Linguistic Processing

1.2.1 Grammar Development

Deep linguistic processing is not a new invention. As mentioned in the previous section, the defining feature of deep linguistic processing is its grammar centric approach. Hand-crafted grammars are complex rule systems developed over decades and encode rich knowledge about the usage of the language. As the results of enduring efforts in grammar engineering, some of the large scale grammars have grown to contain thorough coverage of various linguistic phenomena. While most of the initial attempts in grammar development usually focus on English, more recent progress has taken much broader steps to aim at multilingual grammar engineering. Several projects have initiated collaborative efforts that involve researchers from different institutes around the world who develop grammars for different languages within the same linguistic frameworks using the same development environment.

One of the earlier projects along this line is the LS-GRAM project (Schmidt et al., 1996). Funded by the EU-Commission under LRE (Linguistic Research and Engineering), the LS-GRAM project (January 1994 – July 1996) was concerned with the development of grammatical resources for nine European languages: Danish, Dutch, English, French, German, Greek, Italian, Portuguese, and Spanish. The development was carried out in the framework of the Advanced Language Engineering Platform (`ALEP`). However, the grammars achieved very limited coverage.

The Parallel Grammar Project (`ParGram`, Butt et al. (2002)), aiming at multiple grammar development within the linguistic framework of Lexical Functional Grammar (`LFG`), started in 1994 with three languages: English, French, and German. After more than a decade of development, several of the grammars have grown into broad coverage precision grammars. More than a dozen languages have been added to the project, while great effort has been made to keep the parallelism among the grammars. The development platform of the project, XLE, includes a parser, a generator and a general purpose rewriting system.

With a similar spirit to the ParGram project, the Matrix grammar was developed within the context of the DELPH-IN community. DELPH-IN stands for *deep linguistic processing with HPSG– initiative*, which is a collaborative community with researchers from over a dozen institutions around the world who are working on the implementation and application of linguistic grammars within the framework of Head-Driven Phrase Structure Grammar (HPSG) and the Minimal Recursion Semantics (MRS).

Over the years, several key software components have been developed to facilitate grammar engineering and application. The LKB system is a sophisticated grammar engineering environment that supports grammar development using typed feature structures. Combining with [incr tsdb()] (the competence and performance profiling system), they can be used for treebanking, training and testing statistical models, etc. PET is an efficient HPSG parser that is compatible with the grammars developed with the LKB system. Implemented in C/C++ with various efficient parsing algorithms, the PET parser is of industrial strength and capable of delivering deep linguistic processing techniques to real applications.

Apart from the processing software components, the grammar resources in DELPH-IN are also growing. The Matrix project is a framework for the development of broad-coverage precision grammars for diverse languages in the linguistic framework of HPSG using DELPH-IN software repositories. Initially enlightened by experience with three large scale HPSG grammars developed during the Verb*mobil* project (i.e., ERG for English, JaCY for Japanese, GG for German), the project aims to build a skeleton grammar that extracts the components that are common across these grammars. Such a grammar helps accelerate the development of new HPSG grammars for other languages. It also helps achieve better parallelism between existing grammars. More recently, the Matrix grammar has allowed customization via a web interface. By answering a series of questions regarding the characteristics of the target language (i.e., word order, negation, coordination, etc.), a customized `Matrix` grammar is automatically generated. Under the umbrella of `Matrix`, grammars for about 10 languages are under development. While the new grammars are reaching moderate coverage, the three initial grammars have continued to grow after the Verb*mobil* project, and all reached thorough coverage of basic language phenomena with comparable size. These grammars have since been used in various applications, e.g., machine translation, automatic email response, information extraction, etc.

As mentioned in the previous section, a different approach to grammar development in recent years is based on grammar induction from annotated corpora. These studies have generated interesting linguistic resources (typically in the form of converted and/or enriched annotations from existing treebanks), but were usually limited to a very small number of languages (typically for English only). The most recent noteworthy work includes the extracted `LTAG`s from Penn Treebanks (Xia et al., 2001); the `CCGbank` and derived grammars, a translation of the Penn Treebank into a corpus of Combinatory Categorial Grammar derivations paired with word-word dependencies that approximate the underlying predicate-argument structure (Hockenmaier and Steedman, 2005; Hockenmaier, 2006); and the `HPSG` treebank and induced grammar from Tokyo University, a translation of the Penn Treebank into `HPSG` derivations with predicate-argument structures (Miyao et al., 2004).

For the reasons described in the previous section, this dissertation focuses on deep linguistic processing with hand-crafted grammars. More specifically, we use the `DELPH-IN` resources for most of the experiments in this dissertation. But many of the discussions and conclusions should generalize to other frameworks, as well.

1.2.2 The *"Trinity"* in Deep Linguistic Processing

Given the long existence and promising appearance of deep linguistic processing, it is surprising to realize that, in practice, DLP has been a disfavored approach in real applications for a long time. The experience of past attempts in using deep processing techniques shows that the situation is not completely unjustified. Generally, there are three major challenges in the universe of deep linguistic processing: efficiency, specificity, and coverage.

The efficiency problem concerns the computational complexity of deep linguistic processing systems. Due to the need for thorough modeling of the language, rich mathematical formalisms are normally required. Being more expressive, such formalisms are normally poor in terms of computational tractability. For instance, the time complexity of parsing with unification-based grammars is exponential to the input length. Even if the formalism maintains good theoretical computational complexity, large-scale grammars almost inevitably encounter a huge search space. Fortunately, the problem has been dampened by many new efficient processing techniques. Also, Moore's Law indicates a bright future for deep processing systems with the help of better computer hardware.

The specificity problem is concerned with the preciseness of deep linguistic processing systems. Due to the ambiguous nature of human languages, hand-crafted deep grammars are usually capable of producing a large number of analyses according to linguistic principles. Linguistically sound analyses are not all equally plausible or interesting. Therefore, extra mechanisms are required to model preferences among grammar analyses. Such disambiguation models work with either heuristic preference rules, or statistical ranking models trained on disambiguated grammar outputs. The latter approach has been widely adopted in recent development, and is the de facto standard technique to solve the specificity problem within many frameworks.

The coverage problem concerns the completeness of the grammar description relative to language use. This is the most serious challenge for the deep processing systems to date. While the state of the art broad-coverage precision grammars cover most of the frequent language phenomena, Zipf's law indicates that there are still many uncovered phenomena (e.g., infrequent words, multiword expressions, etc.) in the long tail of the skewed distribution. Also, the static rule systems are not able to account for the evolution of the language. Although continuous grammar development can lead to improvement of the coverage, full coverage can only be achieved with extra mechanisms.

All of the three problems have been studied for a long time. While improvements have been seen in the first two problems, the coverage problem is significantly lagging behind. In this dissertation, I will investigate a closely related but much more general problem of deep linguistic processing: robustness. Its relation to the aforementioned three problems (especially the coverage problem) will be discussed in general in the next chapter, and in detail throughout the rest of the dissertation. The objective is to develop a series of robust processing techniques that will bring the state of the art deep linguistic processing to a new stage where balanced efficiency, specificity and coverage are achieved.

1.3 Structure of the Dissertation

This dissertation discusses a group of new techniques related to robust deep linguistic processing. The discussions and experiments are mostly made within the existing `HPSG` grammar framework, but can be applicable to various similar frameworks, as well.

Chapter 2 gives a brief overview of the robustness problem with natural language processing systems. A case study of the robustness and coverage of a broad-coverage accurate `HPSG` grammar is presented.

Chapter 3 presents the techniques to improve the robustness of the deep lexicon by acquiring the lexical information using statistical machine learning methods. The training corpus can be generated automatically with the grammar. Several techniques to improve the performance are also presented and evaluated.

Chapter 4 expands the discussion of extending the lexicon by moving forward to acquire widely existing but poorly covered linguistic phenomena: multiword expressions. By adapting a similar classification model, in combination with some validation techniques, the grammar performance improves further.

Chapter 5 goes on to investigate the correlation between the lexicon and grammar performance. By simulating deep lexical acquisition results at various precision and recall levels, both the accuracy and the coverage of the grammars are evaluated. The results on two grammars of comparable size for different languages leads to the conclusion that a recall-heavy interpretation of the lexical acquisition results should be preferred.

Having discussed the lexical aspect in details, Chapter 6 moves on to improve robustness in grammar construction. Robust partial parsing is proposed and the related statistical disambiguation models are presented. Also the efficiency concerns are discussed in detail.

Chapter 7 concludes by reviewing the robustness techniques discussed in the dissertation, and outlines some areas of future research potential.

2 Robustness: General

The amount of noise which anyone can bear undisturbed stands in inverse proportion to his mental capacity.

— Arthur Schopenhauer *(1788 - 1860)*

2.1 Overview

Generally, *robustness* is the quality of being able to withstand stresses, pressures, or changes in procedure or circumstance. A system, organism or design may be said to be *"robust"* if it is capable of coping well with (sometimes unpredictable) variations in its operating environment with minimal damage, alteration or loss of functionality.

For natural language processing systems, robustness is usually defined as the capability of handling unexpected inputs.[1] This is an especially important issue when the input to the system is produced by humans, and transferred through various noisy channels. Therefore, the input basically contains two types of variance: it is either produced by a human speaker, or introduced by the communication channel. Both of them can be considered as *noise* from the perspective that they somehow impede the system from capturing the true meaning which is meant to be conveyed behind the message.

The *noise* introduced by communication channels is usually related to the technical limitation of specific media or technology. For instance, in speech text processing, transcription errors (either introduced by human transcribers or by speech recognition systems) are inevitable. Also, the punctuation information is usually missing

[1]Other constraints (time, hardware, etc.) are also considered as aspects of measuring robustness, but are not the focus in this dissertation.

from transcribed speech texts, hence making them different to normal written texts. Therefore, the systems handling such inputs need to tolerate both of these kinds of errors. A similar type of noise also occurs with written texts; for instance, OCR errors in scanned documents. From the viewpoint of computational linguistics, these types of noise need to be properly handled by collaborative research and engineering efforts with various other sub-fields (e.g., speech processing, optical character recognition, etc.).

Unlike the communication channel noise, the other major type of noise is produced by humans. Such noise comprises different phenomena which influence the proper understanding of the language. These phenomena are either not sufficiently covered by conventional linguistic study or lack implementation in specific language resources (e.g., deep linguistic grammars).

The noise generated by humans (called errors in some cases) is usually the combined effect of various competence or performance factors. For instance, misspelling, disfluency, and grammatical errors are common types of noise produced by new language learners. However, under specific circumstances, e.g., under time pressure or when tired, speakers with sufficient knowledge about the language may produce similar types of noise, as well.

Another factor that adds complexity to natural language processing is the evolution of the language. Languages change over time. New linguistic phenomena emerge, while outdated ones perish. More noticeable is the change in the lexicon: new words are created everyday, while old ones slowly fade out. Nowadays, the changing pace is accelerated even more by modern information technologies and globalization, so that new words (either newly created, or adopted from foreign languages) can be distributed at incredible speed.

Since natural languages are such open-end systems, it is difficult to define a clear boundary beyond which the variance should be considered as noise. Nonetheless, it is amazing to see that humans are extremely robust to dramatic variance in language use: slight ungrammaticality in spoken language does not impede effective communication among human speakers; new words can be picked up after just seeing them in use only few times, etc.

In contrast to human adaptivity to the variance of language use, existing natural language processing systems typically lack such robustness. In precision grammar-based deep linguistic processing, the central language resource (i.e., the grammar) is usually formulated as a symbolic rule system which is not especially capable of tolerating such variance in at least two aspects.

First, it is not designed to handle language evolution. State of the art grammar frameworks are formulated on well-defined mathematical formalisms. However, they are typically designed for describing static language phenomena. Whenever a new language phenomenon emerges, new changes must be made to the existing grammar. Occasionally, the new phenomenon does not fit in the design of the grammar. Then the evolutionary change in the language may lead to a revolutionary change of the grammar. Although lately there has been a lot of research efforts on better design of grammar frameworks that keep highly generalized linguistic principles apart from specific language phenomena to allow modularized design of the grammar (see Bender et al., 2002), generally it is still very difficult to guarantee the modularity and extensibility of the grammar with the formalism currently in use.

Second, widely adopted grammar frameworks are not designed for graded grammaticality. Shallow grammars and treebank induced grammars usually make no grammaticality judgment, at all. Such grammars are different from precision grammars in that they are not capable of accurately modeling the language. Therefore, they may perform well in some specific tasks (i.e., parsing) and fail in others (i.e., generation). Precision grammars generally achieve better accuracy by deliberately restricting the constraints, and produce analyses for grammatical sentences only. However, current frameworks usually imply a binary grammaticality. The boundary of grammaticality is rather arbitrarily set by the grammar writer. Also, there is an asymmetry between the treatment of grammatical and ungrammatical sentences: grammatical sentences receive one or more detailed analyses; ungrammatical sentences are dropped to the ground. Various studies have shown that human speakers usually make a graded judgment of grammaticality (cf. Keller, 2000). Ungrammatical sentences are partial processed by human speakers depending on their grammaticality level.

Both of these points indicate that the current grammar frameworks lack the power to properly model the variance in language use as humans do. This directly leads to a lack of robustness in precision grammar-based natural language processing systems. While more advanced formalisms are desirable in future study, it is still largely unclear how long until a really promising framework emerges that can properly address these problems while remaining implementable. On the practical side, huge efforts have been invested in the development of various language resources (grammars, treebanks, processing software) using current frameworks since the mid-1990s. Although limited by their underlying formalisms, these language resources are still of great value.

In this dissertation, we aim at improving the robustness of precision grammar-based deep linguistic processing systems by incorporating a series of novel *robust processing techniques*. Without changing the base formalism, our techniques work as extra modules upon the existing framework, and significantly improve the grammar performance in a robust deep linguistic processing scenario.

2.2 Robustness and Coverage

The robustness problem is closely related to several performance aspects of a deep linguistic processing system. For instance, in some systems, unpredicted input might result to an exhaustive search over the entire solution space, which leads to inefficiency (see Section 6.4 for one example of such a problem and its solution). On the other hand, robust processing can also lead to a specificity challenge. For instance, the extension of the lexicon leads to greater lexical ambiguity, and potentially larger numbers of analyses per sentence.

The other, more directly related, performance aspect of deep linguistic processing system is coverage. In the traditional definition, the coverage of a deep linguistic processing system is the proportion of inputs which i) are well-formed, and ii) receive at least one correct analysis from the grammar. While similar to the definition of robustness, the prerequisite that the input must be well-formed makes the concept of coverage slightly different.

On a closer look, it can be easily realized that the exact meaning of coverage is crucially decided by how well-formedness is defined. In many cases, it is not related to the actual regularity of the input, but rather defined according to application tasks. For instance, a grammar designed for situated dialog systems may consider newspaper texts not well-formed, even though the latter normally have better grammaticality and fluency.

For broad-coverage precision grammars, the coverage is usually measured over a set of carefully chosen test items. The well-formedness is marked (as binary decisions) by grammar writers. Strictly speaking, such coverage tests do not reflect the true robustness of the grammar. High coverage does not automatically imply good robustness.

In this dissertation, we try to use a different measure of coverage in an attempt to properly reflect the robustness of the grammar with the same measure. Rather than relying on a carefully selected test set and manually assigned well-formedness, we rely on large balanced corpora with trusted grammaticality. More specifically, we use the British National Corpus (BNC; Burnard (2000)) for a coverage test of the English Resource Grammar (ERG; Flickinger (2002)). A large number of sentences were selected to create a sub-corpus from the written component of the BNC. The coverage is defined as the proportion of sentences that received at least one reading. Since the BNC is a carefully chosen collection of data with substantial variation while still maintaining good balance, the coverage number is representative of the robustness of the grammar when faced with real running texts. In the next section, we will start with a case study of a coverage test of the ERG with BNC.

2.3 Case Study

This dissertation aims at robust deep linguistic processing with precision grammars. As a starting point, in this section we show a case study of the robustness of a large scale HPSG grammar for English which represents the state of the art grammar engineering outcomes.

The LinGO English Resource Grammar (ERG; Flickinger (2002)) is a broad-coverage, linguistically precise HPSG-based grammar of English. Initially started in 1994, the grammar has undergone continuous development for over a decade, with about 18 person-years of work (as of summer 2007). Since its first application in the Verb*mobil* spoken language machine translation project, the grammar has grown to a precision grammar with reasonably good coverage over unseen running texts. The grammar is now semantically grounded in Minimal Recursion Semantics (MRS; Copestake et al. (1999)). It is developed with the LKB system, and can also be used with the PET parser. The *jun-04* version of the ERG contains 23.6K lines of code in TDL[2] (excluding the lexicon) with about 5K lines of comments. The lexicon contains 12,347 lexical entries, categorized into 728 leaf lexical types. There are in total over 3.5K types defined in the grammar, and about 110 construction rules.

Baldwin et al. (2004) reported an evaluation of the English Resource Grammar with a set of 20K sentences randomly selected from the BNC, where both coverage and accuracy of the grammar were analyzed. Of the utterances with full lexical span (at least one lexical entry exists for each token in the input), the causes of parsing failure are classified into six categories: missing lexical entries, missing constructions, preprocessor errors, fragments, parser failures, and garbage strings.

As mentioned in the last section, to obtain a reliable test set that properly reflects the variations in language use, we need to build from BNC a large sub-corpus. Considering that the recent grammar

[2]TDL (Krieger and Schäfer, 1994), standing for *Type Description Language*, is the formalism foundation based on which several grammar engineering/processing platforms using typed feature structures (including PAGE, LKB, PET, SProUT) are built.

development is aimed at written text processing, we use the written component of the BNC corpus to extract the sub-corpus. More specifically, we used the following heuristic rules to collect the utterances for the sub-corpus:

1. The utterance must have a proper final punctuation (i.e., full stop, question mark, exclamation, etc.), so that the utterance is more likely to match the "well-formed" utterances covered by the grammar;

2. The utterance must not contain non-ASCII characters, so that non-English (sub)utterances or foreign words are not involved;

3. The utterance must not contain more than 20 words, so that the huge amount of data can be deeply processed within reasonable time.

It should be noted that the third requirement is specified from the engineering point of view. Due to the large number of utterances in the corpus, and the exponential complexity (relative to the input length) of unification-based parsing, we select the upper-bound limit of 20 words for our sub-corpus. Empirically, we have found that our current parser handles short utterances fairly efficiently, with an average speed of 1 utterance a second, and a much lower median speed of around 0.2 seconds per utterance. For longer sentences, the parser throughput drops significantly. By setting the 20-word limit (as well as other constraints), we obtain a sub-corpus with a total of 1.8M utterances. Using PCs with 3GHz CPUs, we are able to parse the entire sub-corpus within 4∼5 CPU days.

It is true that there is a strong empirical correlation between the input utterance length and the grammar coverage: the coverage drops with the increase in the utterance length. Therefore, setting a upper bound length limit on the utterance introduces a bias on the absolute figure of the coverage. But in hope that the same data set provides a relative measure for different versions of the grammar, the coverage numbers should be comparable, and reflect the robustness differences between versions.

Using the *jun-04* version of the ERG, we parse the entire BNC sub-corpus with the PET parser. Since we are not interested in the exact outcome of the analysis, we run the parser in the recognizer mode, i.e., checking whether the input utterance has at least one analysis. In this mode, the parser does not need to extract any reading from the parse forest. When using subsumption-based local ambiguity packing, the packed parse forest can be created in practically polynomial time.

As the outcome of the parser, we have one of the following 4 states for each input utterance:

- **P** means that the utterance receives at least one full analysis
- **L** means that the utterance contains at least one lexical gap (input tokens that that do not correspond to any existing lexical entry in the lexicon of the grammar)
- **N** means that the utterance contains no lexical gap but still receives no parse
- **E** for all other cases (e.g., parser crash, timeout or out-of-memory errors)

The overall coverage of the grammar C_{ALL} is estimated as the proportion of utterances in the BNC sub-corpus that is marked as **P** by the parser. The relative coverage with no lexical gap C_{NG} is defined as $\frac{|P|}{|P|+|N|}$ to separate the construction coverage from the lexical coverage. The full lexical span rate R_{FLS} is defined as $\frac{|P|+|N|}{|P|+|N|+|L|}$, which crudely shows the lexical coverage of the grammar.[3] The results are summarized in Table 2.1.

From the results we see that this specific version of ERG contains at least one lexical gap for about 70% of the utterances, setting a miserable upper bound for grammar coverage. This is essentially consistent to the results obtained by Baldwin et al. (2004) (where 32%

[3]It should be noted that the missing lexical entries do not necessarily lead to lexical gap(s) during parsing. Details about lexical gap and lexical coverage is discussed in Section 3.3.

	$\lvert P\rvert$	$\lvert N\rvert$	$\lvert L\rvert$	$\lvert E\rvert$	R_{FLS}	C_{ALL}	C_{NG}
ERG (*jan-04*)	301,503	239,272	1,260,404	96	30.02%	16.74%	55.75%

Table 2.1: Coverage test result of ERG (*jun-04*) on the BNC sub-corpus

of the utterances from their test set are lexical gap free). The 55.75% no-gap coverage shows that even without lexical gaps, the grammar still suffers from a high parsing failures ratio. Baldwin et al. (2004) reported that 40% out of these no-gap parsing failures are caused by missing lexical entries, while 39% are related to missing constructions. We manually evaluated a small subset of the parsing failures from our BNC sub-corpus, and observed a slightly higher proportion of the two error types, both at around 46% of all the no-gap failures. We see the difference as a result of using the aforementioned selection criteria to create our BNC sub-corpus, so that the proportion of garbage strings and parser failures reduced significantly.

Both our experiment and that of Baldwin et al. (2004) arrive at the conclusion that the major robustness barriers for the ERG grammar (and possibly other grammar resources of a similar kind and scale) are from two aspects. First, current static manually compiled lexicons of the grammars are far from sufficient. Second, even with a perfect lexicon, extra robust processing mechanisms are required to handle variations in the construction.

2.4 General Approach

The purpose of robust deep linguistic processing is to ensure that appropriate and meaningful structures are assigned to the input utterances. The processing should preserve the accuracy of the analyses, meaning that the robust output should be well supported by the linguistic resource and underlying theory. Also, the outcome of the robust processing should maintain detailed linguistic information, so that it can still be differentiated from shallow processing outputs.

As mentioned in Chapter 1, there has been quite a lot of previous work on robust processing. For instance, some of the approaches devise various recovery strategies from errors and modify the inputs (Lyon, 1974; Aho et al., 1986; Lehman, 1989; Hipp, 1992; Weng, 1993; Rosé and Lavie, 1997), while others try to extend grammar frameworks that allow the recording and measuring of the grammaticality of the inputs (Fouvry, 2003b).

In order to improve the robustness of the existing grammar resources, in this dissertation we take an evolutionary approach to develop a sequence of robust processing techniques that can be applied to many deep linguistic processing systems as incremental add-on modules.

To start with, we will first look at techniques to automatically acquire linguistic lexical knowledge in Chapter 3, followed by an extension to acquire multiword expressions in Chapter 4. The relation between the lexicon and the grammar performance is discussed in Chapter 5. In Chapter 6, a partial deep parsing strategy is proposed to handle missing constructions.

3 Deep Lexical Acquisition

Order and simplification are the first steps toward mastery of a subject — the actual enemy is the unknown.

— Thomas Mann, *"The Magic Mountain, 1924"*

Deep linguistic processing delivers fine-grained syntactic and semantic analyses which are desirable for advanced NLP applications. However, the limited coverage poses a major barrier. This chapter will discuss automatic deep lexical acquisition techniques which can effectively enhance lexical coverage.

3.1 Motivation

This dissertation is about robust processing techniques with precision deep grammars. As shown in the previous chapter, the lack of robustness and coverage in deep processing techniques is a combined effect of multiple factors. The lack of lexical coverage is one of the major stumbling blocks for deep processing to achieve broad coverage and high robustness.

Take the coverage test result of `ERG` (*jun-04*) on `BNC`, for example. The grammar has at least one lexical gap on 70% of the inputs. The lexical coverage is much lower than the grammar construction coverage (since the same grammar achieves 56% coverage on sentences without a lexical gap). In the traditional pipeline processing model, whenever a lexical gap is found in the input sequence, the processing is aborted. Obviously, this is a weak link in deep processing in terms of robustness. For deep processing, the detailed linguistic information must be provided by the lexical entries. Manually compiling a large scale deep lexicon is a time consuming task. Also, the static lexicon inevitably becomes insufficient due to the evolution of the language. Moreover, in real applications, the noisy inputs are also a robustness challenge for the deep lexicon. A typical type of noise in written text comes from misspellings. Misspelled words are surely beyond the coverage of the lexicon. Whether they can be treated properly is crucial for deep processing in applications.

All these issues indicate the need for (semi-)automatic ways of acquiring lexical information.

3.2 The Lexicon

Lexicon has always been playing a significant role, not just in deep processing, but generally in the entire study of human languages. This is simply because, no matter how complicated and divergent they may be, human languages are all built on some basic units which carry meanings and phonetic values. Without the knowledge of such basic units, language processing is literally impossible. The inventory or knowledge base of these language units is called lexicon, and has played an important role in the study of human languages.

There is no definite answer to the question of what specific information should be encoded into the lexicon, and this is far beyond the scope of this dissertation. Rather, here I will restrict the discussion about the lexicon to the scope of deep processing with precision grammars, i.e., the `DELPH-IN` `HPSG`s.

3.2.1 Lexicon: A Functional View

Deep grammars are usually composed of two main components: i) a lexicon with information about words, and ii) a rule system (usually referred to as grammar, too, but in a more restricted sense) that incorporates the word information into processing.

Different approaches see different boundaries between the lexicon and the rule system. In some highly lexicalized formalisms like `LTAG` and various CGs, the rule system is extremely simplified and abstracted, so that almost all the knowledge about the language goes into the lexicon.

Here we are more interested in seeing the relation between different grammar components and robustness. Therefore, in this dissertation, we will restrict the discussion on the formalisms and grammars to the points where the distinction between grammar rules and lexicon is apparent. More specifically, the lexicon L in this discussion is essentially a function which maps each lexeme $w_i \in W$ to a subset of abstract symbols $A = \{a_1, a_2, \ldots, a_m\}$:

$$L : W \to \mathcal{P}(A). \tag{3.1}$$

It is with this set of abstract symbols that the information about the lexemes is recorded. The polymorphic nature of lexemes is captured by mapping a single word/lexeme onto multiple abstract symbols. The remaining part of the grammar, namely the grammar rules, generalizes the generative machinery as well as various linguistic restrictions of the language. The rules interact directly with the abstract symbols, not the words, to keep the generality of the linguistic principles. Despite the different formalisms, the information encoded in such an abstract symbol set is usually atomic, meaning that information encoded is self-sustaining. More details will be discussed in Section 3.2.2. In such a view, the role of the lexicon in the grammar is essentially the "joint" between the list of words/lexemes and the rule system.

It should be noted here, that the functional view of the lexicon here is mainly aimed at the parsing tasks where the inputs are words and the outputs are analytical structures. However, this view does not conflict with other applications of the grammar. For example, in generation tasks, the inputs are semantic representations and the outputs are word sequences. In such cases, the lexicon serves as another function which maps from the semantic units (i.e., predicates) $s_i \in S$ to sets of words/lexemes $w_i \in W$:

$$L' : S \to \mathcal{P}(W). \tag{3.2}$$

The two mappings of the lexicon also enable us to separate the syntactic and semantic layers. Mapping L mainly deals with the syntactic restrictions of the word, while L' maps the concept units to words. Although L' is also potentially interesting, especially for the study of lexical semantics, this dissertation focuses more on the syntactic processing and its relation to the lexicon. If not mentioned otherwise, the mapping L will be used as the default definition of lexicon hereafter.

3.2.2 Atomic Lexical Types

As mentioned earlier, the grammar rules interact directly with a set of abstract symbols, instead of concrete words/lexemes, and the lexicon creates the correspondence between words/lexemes and the abstract symbols. Therefore the abstract symbols must carry the complete lexical information, so that the syntactic characteristics of different words can be properly differentiated and handled.

It has been well studied that lexical information for human language is knowledge-rich. Different formalisms have developed different methods for knowledge representation of lexical information. For example, in `HPSG`, the feature structure in Figure 3.1 will be used as a brief description of the lexical information carried by the proper name *"Mary"*.

The lexical information encoded in this structure includes the word category, word stem/lexeme, valency information, person, number, gender, and also semantic constraints. In principle, such structures can be used to encode the lexicon, since they provide all the information. However, the direct encoding approach is not an optimal way either from the theoretical or the practical point of view. From the theoretical point of view, linguistic generality will be sacrificed in such an approach. The abstract symbol set is the vocabulary of the grammar rules. If the linguistic information is specified completely at the lexical entry level, then it will be difficult to generalize the linguistic phenomena over groups of words, and the role of word category becomes less significant. Practically, this approach is not optimal

$$
{}_{noun}\begin{bmatrix}
\text{STEM} & \left\langle \text{“M\textsc{ary}”} \right\rangle \\
\text{SYNSEM} & {}_{synsem}\begin{bmatrix}
\text{LOC} & {}_{local}\begin{bmatrix}
\text{CAT} & {}_{cat}\begin{bmatrix}
\text{HEAD} & {}_{noun}\begin{bmatrix} \text{CASE} & \textit{case} \\ \text{PRD} & \textit{bool} \end{bmatrix} \\
\text{VAL} & {}_{valence}\begin{bmatrix} \text{SUBJ} & \langle\rangle \\ \text{SPR} & \langle\rangle \\ \text{COMPS} & \langle\rangle \end{bmatrix}
\end{bmatrix} \\
\text{CONT} & {}_{nom\text{-}obj}\begin{bmatrix}
\text{IND} & \boxed{1}\ {}_{ref}\begin{bmatrix} \text{PER} & \textit{3rd} \\ \text{NUM} & \textit{sg} \\ \text{GEN} & \textit{fem} \end{bmatrix} \\
\text{RESTRS} & \left\langle {}_{naming}\begin{bmatrix} \text{NAME} & \textit{Mary} \\ \text{BEARER} & \boxed{1} \end{bmatrix} \right\rangle
\end{bmatrix} \\
\text{CONX} \mid \text{BKGRD} & \left\langle {}_{female}\begin{bmatrix} \text{INST} & \boxed{1} \end{bmatrix} \right\rangle
\end{bmatrix} \\
\text{NONLOC} & \textit{nonlocal}
\end{bmatrix}
\end{bmatrix}
$$

Figure 3.1: Lexical information for *“Mary”* in a Typed Feature Structure

either, for the lexicographer must provide the complete description for each lexical entry, even though a large part of such information is shared among many different words.

Therefore there is always some level of abstraction for the construction of the lexicon. Again, we take HPSG as an example. The lexicon in HPSG is usually organized into a *type hierarchy*, which is a multiple inheritance system. The super-types contain fewer constraints and are more general. The sub-types monotonically inherit all the constraints from their parents, and optionally introduce some extra constraints, therefore become more specific. The inheritance type system allows multi-layers of encapsulation of lexical information. For the completeness of lexical description, usually only the leaf types are assigned for each lexical entry. The leaf types are those maximum types on the hierarchy which do not have any sub-types.

With the type system, the actual lexical entries look like the following[1]:

```
dog_n1 := n_-_c_le &
 [ STEM < "dog" >,
   SYNSEM [ LKEYS.KEYREL.PRED "_dog_n_1_rel",
            PHON.ONSET con ] ].

dog_v1 := v_np_le &
 [ STEM < "dog" >,
   SYNSEM [ LKEYS.KEYREL.PRED "_dog_v_1_rel",
            PHON.ONSET con ] ].

blue_a1 := aj_-_i_le &
 [ STEM < "blue" >,
   SYNSEM [ LKEYS.KEYREL.PRED "_blue_a_1_rel",
            PHON.ONSET con ] ].

blue_n1 := n_-_mc-col_le &
 [ STEM < "blue" >,
```

[1]Examples are taken from the ERG (*nov-06*).

```
   SYNSEM [ LKEYS.KEYREL.PRED "_blue_n_1_rel",
            PHON.ONSET con ] ].

give_in_v1 := v_p_le &
 [ STEM < "give" >,
   SYNSEM [ LKEYS [ --COMPKEY _in_p_sel_rel,
                    KEYREL.PRED "_give_v_in_rel" ],
            PHON.ONSET con ] ].

give_up_v1 := v_p-np_le &
 [ STEM < "give" >,
   SYNSEM [ LKEYS [ --COMPKEY _up_p_sel_rel,
                    KEYREL.PRED "_give_v_up_rel" ],
            PHON.ONSET con ] ].
```

The entries listed above are separated by blank lines. At the beginning of each entry, the name of the lexical entry is given (i.e., "dog_n1", "dog_v1", ...). Then a unique leaf lexical type assigned to the entry (i.e., *"n_-_c_le"*, *"v_np_le"*, etc.). The rest of the entry defines some extra feature constraints. This part of information is largely word-specific, i.e., the "stem" of the word and the semantic relation it introduces. The mapping from lexemes to the possible feature structures is straightforward and efficient. It also allows a word/lexeme mapping to different feature structures via multiple lexical entries with the same "stem" feature. The different linguistic behaviours (even if subtle) can be captured by the corresponding lexical types.

A closer look at the type hierarchy helps clear up more of the mystery of the lexical types. Figure 3.2 is a small fragment of the ERG lexical type hierarchy[2]. The types above are the more general types (super-types), while the types below are the more specific types (sub-types). The types at the bottom of the hierarchy are called *"le"*

[2]This fragment of lexical type hierarchy is taken from the ERG (*jun-04*). The naming of the lexical types has undergone significant change since then. Also, the complete lexical type hierarchy is much more complex with lots of multiple inheritance.

types (because they have names which end with *"le"*). These are the leaf lexical types that will be assigned to lexical entries.

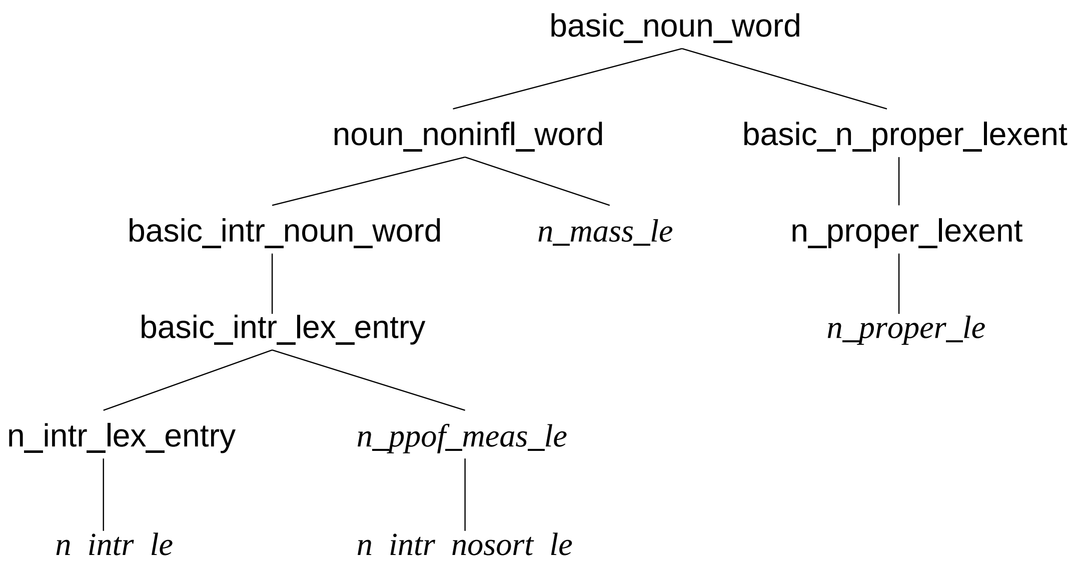


Figure 3.2: A fragment of the `ERG` lexical type hierarchy for nouns

Among many interesting features of a hierarchical lexicon, the leaf lexical types provide two important characteristics that are crucial for this discussion.

The first important characteristic is that the *"le"* types are maximum types, therefore carry almost a complete description of the lexical information. The complete description would, in addition, include the word specific information like the "stem" and the semantic relation. Also, in some cases, the *"le"* types require extra feature constraints to complete the description. For example, for the verb particle constructions, the *"le"* type *"v_p_le"* acquires the particle information via the value of attribute *"--COMPKEY"*. Such an exception can be addressed by compiling out all the possible values of such attributes as sub-types. Large grammar engineering results have shown that the possible values for such attributes on the *"le"* types are usually limited. The resulting types will have the *full specification* of the lexical information.

The other important characteristic is *independence*. There is no direct interaction between the lexical types. The design and application of *"le"* are not directed affected by other types.

To make the discussion a little more general, we will use the term *atomic lexical types* henceforth to refer to any of the abstract symbols which encapsulate the lexical information and have the above two characteristics.

Typical large scale deep grammars usually have rather complete sets of atomic lexical types. The problem of lack of lexical coverage is normally caused by insufficient mapping from lexemes to the appropriate atomic lexical types, rather than missing atomic lexical types.[3] Therefore, the task of lexical acquisition is to assign the appropriate types for the words or lexemes and create the lexical entries.

The task of lexical acquisition is especially important and difficult for deep grammars. The grammars heavily rely on fine grained word categories and detailed lexical information. This leads to a large set of possible atomic lexical types. If the lexical information is missing, the robustness and coverage of the deep processing will be hurt directly.

Manual compilation of broad-coverage lexicons for deep grammars is a time consuming task. It has been estimated that for a large `HPSG` like the English Resource Grammar (`ERG`; Flickinger (2000)) with over 800 leaf lexical types, a skilled lexicographer can create about 50 entries per hour. For a middle size lexicon of 30,000 entries, this will take about 15 person-weeks for initial creation, and much longer for validation.

The lexicographers must be familiar with the interaction between atomic lexical types and the grammar rules. They also need to be able to discriminate the subtle difference between different lexical types. Therefore, in many cases, the manual creation of a high quality lexicons for deep grammars is done by the grammar developers themselves, and occasionally also people who are familiar with the internal structure of the grammar under the close supervision of the grammar developers. Thus, large scale distributed annotation is difficult in practice.

Moreover, as the deep grammars themselves are constantly under

[3]Although difficult to prove, the claim is justified (at least for grammars like `ERG`) through the case study of manual lexical extension reported in Section 3.2.3.

evolution, the set of atomic lexical types may also change, making the migration of the lexicon a non-trivial task. Last but not least, there is no guarantee on the quality (i.e., in terms of coverage and accuracy) of the manually compiled lexicon. If the process of deep lexical acquisition can be automated, a lot of manual effort will be saved.

3.2.3 A Case Study of Manual Lexical Extension

To demonstrate the amount of work required for manual lexical extension, a case study of manual lexical extension has been carried out. The objective of the task was to extend the lexical coverage of the *LinGO English Resource Grammar* (`ERG`) for a web-derived corpus on tourism in Shanghai.[4]

Before the lexical extension, the grammar has full lexical coverage for about 20% of the strings in the corpus, out of which 55% receive a parse. In other words, only 11% of the strings in the corpus are successfully parsed. The lexical coverage is significantly lower than the `BNC` road-testing because there are a lot of transliterated Chinese proper names and idioms.

The manual extension of the lexicon consists essentially of:

1. Discovering new word use/multiword expressions;
2. Mapping the new words to one of the leaf lexical types in the `ERG` lexical hierarchy.

The extension is done only for unknown words. Words already in the lexicon but with missing entries are not considered. The manually extended lexicon contains 1,575 new entries. Most of them are nouns, adjectives and adverbs. The distribution of new noun entries over different lexical types are shown in Table 3.1.

[4]The corpus contains 1,600 strings of texts on tourism in Shanghai, written in English and mostly by non-native English speakers. Average sentence length is 18 words. The corpus is analogous to the `Rondane` corpus built by Becky Neil for the `LOGON` project (Oepen et al., 2004) in Norway.

Lexical Types	Number of Entries
n_proper_le	831
n_intr_le	386
n_mass_le	94
n_mass_ppcomp_le	49
n_ppof_meas_le	6
n_proper_city_le	5
n_plur_le	4
n_ppof_le	3
n_proper_abb_le	2

Table 3.1: Distribution of noun entries in a manually extended lexicon

It is clear that the distribution of different new word types is extremely uneven, with proper names more likely to be missing in open text processing.

Using the extended lexicon, the ERG is able to parse about 53% of the strings in the corpus, 4.8 times more than before. This confirms that lexicon extension is indeed an important aspect towards broad coverage deep processing.

The case study also shows that manual extension is time consuming. It takes an annotator about five workdays to read through the corpus and decide the lexical types of the new words with the aid of some scripts. Double checking of the lexicon requires even more effort. A large subset of this manually extended lexicon has been incorporated into a recent official ERG release (since version *apr-05*).

The remainder of the chapter is dedicated to the technical details of automated deep lexical acquisition.

3.3 Lexical Error Detection

3.3.1 Grammar Errors and Lexical Errors

Generally speaking, the inadequacy of deep grammar for processing tasks (either parsing or generation) is due to grammar errors.

Roughly, the errors can be classified into two categories: overgenerating errors and undergenerating errors.

As the name suggests, the overgenerating errors will lead to unsupported (usually inappropriate) analyses, and reduce the accuracy of the grammar. The undergenerating errors will make the grammar not be able to cover some of the "correct" analyses, therefore reduce the coverage and robustness of the grammar.

The errors with the deep lexicon can be accordingly classified into two categories as well. When a lexical entry is missing from the lexicon, it will prevent sentences containing the corresponding use of the word/lexeme from being either parsed or generated, thus leading to undergeneration.

On the other hand, when an erroneous lexical entry enters the lexicon, it will potentially allow erroneous analyses to be generated by the grammar, which leads to overgeneration. In some cases, the damage caused by an erroneous lexical entry can be even worse. Introducing erroneous entries on frequent words or with functional lexical types may also lead to significant degrading of the processing efficiency. I will come back to this point later in Chapter 5.

It should be noted that the effect of missing lexical entries on the grammar coverage is non-linear. Some simple calculations might help to clarify this point. Suppose the average sentence length is n words. And the average entry missing rate for each word instance is p_{le}. Then on average there will be $1 - (1 - p_{le})^n$ of the sentences which fail to produce the correct reading because of the missing lexical entry(s). It is clear that the coverage will change exponentially in relation to the lexical coverage. A more detailed analysis of the relation between lexicon quality and grammar performance will be discussed in Chapter 5.

For the purpose of robust processing, we want to start with a relative small but accurate core lexicon in order to: i) overcome the lexical gaps by automatically detecting the words/lexemes which do not have proper lexical entries in the core lexicon; ii) generate new lexical entries for these word/lexemes by assigning appropriate atomic lexical types to them, while avoiding generating harmful erroneous entries.

3.3.2 Lexical Gaps

The first step of automated deep lexical acquisition is to discover a list of words/lexemes which are missing lexical entries. At a first glance, this might seem to be an easy task. Naively we can use the words/lexemes from a large raw text set and check their availability in the lexicon. Whenever a word/lexeme has no corresponding entry in the lexicon, new entries need to be acquired for it. This easy technology can be even reinforced by morphological analysis over the lexemes.

Let us take the `DELPH-IN` processing architecture as an example. For the parsing task, the input sequence first goes through a preprocessing module, which is a finite state automaton-based rule system that tokenizes the input sequence and filters out the noise. Then these tokens (also called input items) are fed into the lexical parser. The lexical rules are applied to these input tokens (usually for morphological analysis), and the appropriate lexical entries are activated according to the result of the lexical analysis (i.e., via the matching of "stem" attribute). Activated lexical entries, combined with the information from the input items, instantiate the lexical items on the parsing chart. And the (non-lexical) grammatical parsing starts after the lexical parser has created all the lexical items on the parsing chart.

Whenever there is an input item that does not have any corresponding lexical entry, the lexical parser fails to create any lexical item for it. This finally leads to a *lexical gap* which means an interval on the parsing chart upon which there is no lexical item created. Therefore the existence of lexical gaps is a direct indication of missing lexical entries for the input token(s) within the gap.

3.3.3 Error Mining

Detection of lexical gaps is an easy yet efficient way of locating unknown words. However, there are other cases when the word/lexeme does have one or more entries in the lexicon, but none of these entries are good/correct for the specific context. The existing entry might fill in the lexical gaps, but is not able to provide correct lexical information in the later parsing stages.

Such a lexical error is much harder to detect, even for human lexicographers. It is often the case, that without enough data showing variation in the use of the word, the human lexicographer tends to lose some less frequent usage of the words. What a human lexicographer usually does in order to avoid such a mistake is to consult with large corpora and see whether there are uncovered usages of the given word.

For the automated lexical acquisition, we apply a similar idea of discovering lexical errors with large corpora. The technique we use is called "error mining" (van Noord, 2004). It has been introduced and successfully used in the development of the Dutch `Alpino` deep parsing system (Bouma et al., 2001).

The basic idea behind "error mining" is to test the grammar performance over large corpora. The performance change over different input combinations is used as an indication of "error".

More specifically, the "error mining" techniques proposed by van Noord (2004) define the concept of *parsability* as the following:

$$R(w_i \ldots w_j) = \frac{C(w_i \ldots w_j, OK)}{C(w_i \ldots w_j)} \tag{3.3}$$

where $C(w_i \ldots w_j)$ is the number of sentences in which the n-gram $w_i \ldots w_j$ occurs, and $C(w_i \ldots w_j, OK)$ is the number of sentences with a successful parse which contains the n-gram. Since the length of the n-gram can be arbitrary, a longer sequence $w_h \ldots w_i \ldots w_j \ldots w_k$ is only considered when its parsability is lower than the parsability of each of its sub-sequences:

$$R(w_h \ldots w_i \ldots w_j \ldots w_k) < R(w_i \ldots w_j) \tag{3.4}$$

.

If the parsability of a given n-gram is significantly lower than others, it means that the grammar is more likely to fail in analyzing the sentences containing the n-gram. It is a clear indication that the grammar is having difficulty handling the n-gram. And probably there are grammar errors related to the n-gram.

When applied to the development of the parsing systems, this technique is very general and efficient in finding different types of grammar errors. As reported by van Noord (2004), it helped to efficiently discover various errors for the `Alpino` system, including errors in tokenization, mistakes in lexicon, incomplete lexical descriptions, frozen expressions with idiosyncratic syntax, incomplete grammar description, etc. However, van Noord (2004) did not provide more insights on any systematic ways of analyzing low parsability n-grams. The n-grams were investigated and classified manually. Therefore the entire process of error discovery is not fully automatic.

In order to investigate the applicability of error mining in missing lexical entry detection, experiments on the English Resource Grammar with the British National Corpus were carried out. With the same environment setup as the coverage test in the previous chapter, we calculated the parsability for the n-grams (with a minimum frequency of 5) in the data set from `BNC` with arbitrary n. All the n-grams with parsability lower than 0.1 were recorded for further investigation. For the English Resource Grammar version *jan-06*, 3826 low parsability n-grams were recorded. Among them, 34% are unigrams, 48% are bi-grams, and the rest are trigrams or above (see Table 3.2).

	#	%
Unigram	1287	34%
Bigram	1852	48%
Trigram+	687	18%

Table 3.2: Distribution of low parsability n-grams ($R < 0.1$) discovered by error mining with `ERG` (*jan-06*) and `BNC`.

After consulting with the grammar developers, we found that most of the uni-grams are directly related to their missing lexical entries. Therefore, with some simple filtering heuristics, we are able to get a candidate list of words that need lexical extension, together with the contexts in which they currently failed to parse.

It should be noted here that there are cases where a missing lexical entry is related to n-grams with $n > 1$. This is typical for the missing entries for multiword expressions. We will leave the detailed discussion of how to detect and acquire `MWE` entries for the next chapter.

3.4 Acquiring Lexical Entries

With a list of candidate words (and their context) for lexical extension, the next step of deep lexical acquisition is to generate appropriate entries for the candidate words.

3.4.1 Related Work

In recent years, some approaches have been developed to automatically acquire new lexical entries in linguistic grammars. Such approaches can be broadly categorized as either symbolic or statistical.

Symbolic Approaches

The symbolic approach assumes that the words which are missing lexical entries are known beforehand. As proposed by Erbach (1990), there are three different stages of processing:

1. The sentence with the unknown word is parsed. There are no special requirements for the parsing algorithm, but the lexical look-up procedure needs to be modified.

2. Based on the syntactic structure of the parse, information about the unknown word can be extracted.

3. The information obtained in step 2 may be too fully specified for a lexical entry. Therefore a filter is applied to it to create a new lexical entry.

In constraint-based parsing, the first step is achieved by using so-called *underspecified lexical entries*, which provide fewer (or no) constraints. Such underspecified lexical entries act like a place-holder, and allow the full parse to be constructed. When the parse is created, various lexical information can be gathered from the constraints applied to it by the grammar rules.

To give an example of how this might work, consider the sentence “kangaroo jumps” with “kangaroo” assumed to be unknown to the grammar lexicon. An underspecified lexical entry is used for “kangaroo”, which only contains the “stem” information:

$$\begin{bmatrix}\textsc{stem} & \left\langle \text{"}\textsc{kangaroo}\text{"} \right\rangle\end{bmatrix}.$$

The underspecified entry will unify with the grammar rule, and lead to a successful analysis of the sentence as shown in Figure 3.3, where the sub-feature-structure $\boxed{1}$ is shared between the lexical item of “kangaroo” and the subject of the sentence. According to the agreement constraints, the person and number features of the subject are identical to the verb via the structure sharing $\boxed{2}$ and $\boxed{3}$. Therefore, the lexical information of third person singular is propagated backward into the lexical entry for “kangaroo”.

The main difficulty with such an approach concerns the third step. As Erbach (1990) pointed out, the resulting lexical entry might be too specific based on the single observation of the unknown word. For example, the specific case information of the nouns is not appropriate for the lexical entries as it will disable the use of the lexical entry for the same noun with different cases. Therefore, extra generalization on the resulting entries is required. What Erbach (1990) did not point out is that the approach may also overgeneralize, if the given context does not provide enough lexical information for the unknown word, i.e., the selectional restrictions of verbs and adjectives, predicative

$$\begin{bmatrix} \text{HEAD-DTR} & \begin{bmatrix} \text{STEM} & \left\langle \text{"JUMPS"} \right\rangle \\ \text{HEAD} & {}_{\textit{verb}}\begin{bmatrix} \text{PERSON} & \boxed{2}\ \textit{3rd} \\ \text{NUM} & \boxed{3}\ \textit{sg} \end{bmatrix} \\ \text{SUBJ} & \boxed{1}\ {}_{\textit{noun}}\begin{bmatrix} \text{AGR} & \begin{bmatrix} \text{PERSON} & \boxed{2} \\ \text{NUM} & \boxed{3} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \text{NON-HEAD-DTR} & \boxed{1} \begin{bmatrix} \text{STEM} & \left\langle \text{"KANGAROO"} \right\rangle \end{bmatrix} \end{bmatrix}$$

Figure 3.3: The attribute-value matrix of the parsed sentence with the underspecified lexical entry for the unknown word "kangaroo"

vs. attributive usage of adjectives, case and form of PP arguments, valence class of verbs, etc.

Based on the same idea, Barg and Walther (1998) took an incremental approach to handling unknown words in HPSG. They assigned revisable information to one of two classes, namely *specializable* or *generalizable*. This revisable information would be used to determine which lexical information should be gathered or generalized for the lexical entry. However, Barg and Walther (1998) only provided a working example of the idea. No empirical results were shown. Also, extra modification to the grammar would be required to provide the revisable information.

Fouvry (2003a) followed a similar idea, with more implementation concerns about the technical details. A small scale experimentation was carried out with the LKB and ERG. Although the mechanism worked promisingly on short sentences, the underspecified lexical entries with fewer constraints allow more grammar rules to be applied while parsing, which makes them computationally intractable. It gets even more complicated when two unknown words occur next to each other, potentially allowing almost any constituent to be constructed. Some more experiments have also been carried out with some modification to the grammar rules to reduce the ambiguity. No concrete results have been presented relating to the improvement in grammar performance, either for parsing or for generation.

Statistical Approaches

One of the main reasons that the symbolic approach failed to materialize in applicable techniques for lexical acquisition is that the lexical information was drawn from very little data. This makes the method vulnerable to errors introduced by not seeing representative contexts. The ultimate purpose of lexical acquisition is to enhance the coverage/robustness of deep processing. Its applicability is doubtful if the method lacks robustness itself.

The data-driven approach of handling unknown words has been widely used in part-of-speech (PoS) tagging tasks (see Brill, 1994; Ratnaparkhi, 1996; Brants, 2000). Brill's tagger begins by tagging unknown words as proper nouns if capitalized, common nouns if not. Then the tagger learns various transformational rules by training on a tagged corpus. It applies these rules to unknown words to tag them with the appropriate part-of-speech information. Ratnaparkhi (1996)'s MXPOST encodes the affixes of the unknown words as features and uses them in the maximum entropy scoring model. The TnT tagger uses the conditional probabilities of the tags given a suffix string of the word in the smoothing formula. Theoretically, the task of part-of-speech tagging is very similar to lexical acquisition in our functional view of the lexicon. Unknown words are assigned to different categories. The main difference is that the number of possible word categories in PoS tagging is much smaller as compared to the atomic lexical types in deep grammars. Also the PoS tagger typically assigns every word in the input sequence a tag, while in deep lexical acquisition only the missing words are used for acquiring new entries.

Thede and Harper (1997) reported an empirical approach towards unknown lexical analysis using morphological and syntactic information. The experiment result on the TIMIT corpus showed that the parser performance was greatly enhanced. However, the work was done for a shallow parser with very limited number of word classes.

The applicability to lexicalist deep grammars with lots of lexical types is unknown.

Baldwin (2005a) took a data-driven approach to automated lexical acquisition for deep grammars. Focused on generalizing the method of deriving deep lexical acquisition models on various secondary language resources, Baldwin (2005a) used a large set of binary classifiers to predict whether a given unknown word is of a particular lexical type. This data-driven approach is grammar independent and can be scaled up for large grammars. Evaluation was via *type precision*, *type recall*, *type F-measure*, and *token accuracy*. And the results showed that when different evaluation metrics are used, different conclusions can be drawn from the data.

van de Cruys (2006) took a similar approach over the Dutch `Alpino` grammar (Bouma et al., 2001). Specifically, he proposed a method for lexical acquisition as an extension to automatic parser error detection, based on large amounts of raw text (see van Noord, 2004). A maximum entropy classifier is trained to select the correct lexical categories. The method was evaluated using *type precision*, *type recall* and *type F-measure*. The results show that both morphological information and the parse selection models help to improve the performance of the classifier. However, these numbers fail to give us any insight into the impact of lexical acquisition on the parser performance.

3.4.2 Lexical Acquisition as a Classification Task

In this dissertation, I will also follow the statistical approach and draw on large amounts of linguistically annotated data to help the acquisition of lexical knowledge. However, ideas from symbolic approaches, i.e., underspecification in lexical entries as mentioned in the previous section, are also borrowed.

Specifically, the statistical model we want to build should be able to assign atomic lexical types to the given candidate words. The first design issue we are concerned with here is how many new lexical entries should be created for a given candidate word. The model should be able to allow multiple entries for the same word as a word can potentially have multiple readings or uses.

Baldwin (2005a) used a set of binary classifiers each one corresponding to a specific atomic lexical type. These classifiers decide independently for each candidate word whether it can combine with the corresponding lexical type to make a new lexical entry or not. This design does allow acquisition of multiple entries per word. A potential problem is that the classifiers are independent of each other, therefore for a given candidate word, either none or many lexical entries will be generated. This is an interesting feature, as it might allow us to further investigate the completeness of the atomic lexical types. However, this is not desirable when the robustness of the parser is concerned. Suppose an online robust processing scenario. When the grammar encounters and detects an unknown (or misspelled) word in the input sequence, the lexical type prediction model will be invoked to create new lexical entries for the word. For this specific context, not all the readings and uses are either predictable or useful. And there should be exactly one entry which will allow the grammar to generate the correct/intended reading (under the assumption that there is no intended lexical ambiguity). In this dissertation, the design of the statistical lexical acquisition model aims at maximizing the robustness of the deep processing.

Although deep processing is capable of generating thousands of valid readings per sentence, normally only one of them is really meant by the speaker. The *"true"* reading corresponds to one lexical entry for the unknown word given the context. Hence our statistical model generates one new lexical entry for each instance of the candidate words. This entry will be temporarily active during the processing of the sentence, for the specific input token(s). If necessary, these lexical entries can be merged later and added permanently to the lexicon.

3.4.3 Predictor Based on General Purpose PoS Taggers

As mentioned earlier, the task of lexical type prediction is essentially similar to the task of part-of-speech tagging. To start with a simple model, we will build a prediction model out of a general purpose PoS tagger.

A typical PoS tagger assigns a (unique or ambiguous) *part-of-speech* tag to each token in the input. We use the lexical types as the tagset and train the tagger on the *Deep Linguistic Resource* (*DLR*). Whenever there are unknown words in the input string, we tag the sequence with the tagger and use the output lexical type to create new lexical entries for unknown words.

State-of-the-art PoS taggers can normally achieve precision above 95%, given a large training corpus. Performance of different tagging models differs slightly. We used a *HMM*-based tagger (TnT; Brants (2000)) and a *ME*-based tagger (MXPOST; Ratnaparkhi (1996)) in our experiments.

TnT is a *HMM*-based trigram tagger. The tags $t_1, t_2, \ldots, t_T$ for a given sequence of words $w_1, w_2, \ldots, w_T$ are generated by calculating:

$$\underset{t_1 \ldots t_T}{\operatorname{argmax}} \left[\prod_{i=1}^{T} P(t_i|t_{i-1}, t_{i-2}) P(w_i|t_i) \right] P(t_{T+1}|t_T) \tag{3.5}$$

where t_{-1}, t_0 and t_{T+1} are beginning-of-sequence and ending-of-sequence markers. The transition and output probabilities are estimated from a tagged corpus. Maximum likelihood probabilities $\hat{P}$ from relative frequencies are calculated as:

$$\begin{array}{rl} Unigrams: & \hat{P}(t_3) = \frac{f(t_3)}{N} \\ Bigrams: & \hat{P}(t_3|t_2) = \frac{f(t_2,t_3)}{f(t_2)} \\ Trigrams: & \hat{P}(t_3|t_1,t_2) = \frac{f(t_1,t_2,t_3)}{f(t_1,t_2)} \\ Lexical: & \hat{P}(w_3|t_3) = \frac{f(w_3,t_3)}{f(t_3)} \end{array} \tag{3.6}$$

TnT also includes a linear interpolation smoothing paradigm and a suffix analysis based unknown words handling. For inflected languages like English and German, it works accurately and efficiently.

MXPOST is a Maximum Entropy model based PoS tagger. The probability model p is defined over the context and tags $\mathcal{H} \times \mathcal{T}$ as:

$$p(h,t) = \pi\mu \prod_{j=1}^{k} \alpha_j^{f_j(h,t)} \tag{3.7}$$

where π is a normalization constant, $\mu, \alpha_1, \ldots, \alpha_k$ are model parameters, and $f_1, f_2, \ldots, f_k$ are features such that $f_j(h,t) \in 0, 1$.

Given a sequence of words $w_1, w_2, \ldots, w_n$ and tags $t_1, t_2, \ldots, t_n$, the likelihood of the model p is:

$$L(p) = \prod_{i=1}^{n} p(h_i, t_i) = \prod_{i=1}^{n} \pi\mu \prod_{j=1}^{k} \alpha_j^{f_j(h_i,t_i)} \tag{3.8}$$

It can be shown that when the observed feature expectation $\tilde{E}f_j$ equals the model's feature expectation Ef_j,

$$Ef_j = \tilde{E}f_j \approx \sum_{i=1}^{n} \tilde{p}(h_i)p(t_i|h_i)f_j(h_i,t_i), \qquad 1 \leq j \leq k \tag{3.9}$$

it uniquely maximizes the entropy

$$H(p) = -\sum_{h\in\mathcal{H},t\in\mathcal{T}} p(h,t)\log p(h,t) \tag{3.10}$$

and uniquely maximizes the likelihood $L(p)$. The model parameters are obtained with *Generalized Iterative Scaling* (Darroch and Ratcliff, 1972).

Then the best tag sequence is found by a beam search using the conditional tag probability

$$p(t|h) = \frac{p(h,t)}{\sum_{t'\in T} p(h,t')} \tag{3.11}$$

and assuming tag sequence probability as

$$P(t_1 \ldots t_n | w_1 \ldots w_n) = \prod_{i=1}^{n} p(t_i|h_i) \tag{3.12}$$

With these tagging models, we can train the taggers that assign leaf lexical types to all the words. But only the types assigned to unknown words are used to generate new entries.

To evaluate how the size of a tagset might influence the prediction precision, a comparison is carried out on the ERG with a different tagset (see Section 3.4.7).

3.4.4 Maximum Entropy-Based Classification Model

A potential problem with the general purpose PoS tagger based model is that the tagger tags every word in the input sequence and tries to maximize the overall probability. For the purpose of lexical type prediction, this is neither necessary nor optimal as we only want the prediction output for specific candidate words.

Another problem is that the general purpose PoS taggers are usually designed to handle a relatively small number of PoS tags. In our prediction model, the set of possible atomic lexical types is much larger.

Therefore, we further turn to a more general statistical model. Formally, the model takes as input the context of the candidate word c (including the candidate word w itself), and outputs an atomic lexical type $t \in T$. The conditional log-linear probability model is defined as:

$$p_\Lambda(t|c) = \frac{\exp \sum_{j=1}^{m} \lambda_j f_j(t,c)}{\sum_{t' \in T} \exp \sum_{j=1}^{m} \lambda_j f_j(t',c)} \tag{3.13}$$

where $f_1, \ldots, f_m$ are called "features", and $\Lambda = \lambda_1, \ldots, \lambda_m$ are the parameters. For a given context c, $p_\Lambda(t|c)$ is a proper probability model for all the possible atomic lexical types $t \in T$.

Given a set of training data as pairs of context and type (c_1, t_1), $\ldots$, (c_n, t_n), the parameters $\Lambda = \lambda_1, \ldots, \lambda_m$ are chosen to maximize the (pseudo-)log-likelihood of the training data (according to Johnson et al. (1999) and Malouf (2002)):

$$L(\Lambda) = \sum_{i=1}^{n} \tilde{p}(c_i, t_i) \log(p_\Lambda(t_i|c_i)) \tag{3.14}$$

where $\tilde{p}(c_i, t_i)$ denotes the observed probability of (c_i, t_i) in the training data.

For each context c, we calculate the conditional probability for each possible atomic lexical type, and the type with the highest conditional probability will be selected. Henceforth, I will refer to this maximum entropy based model as the lexical type prediction model, or simply as the lexical type predictor.

3.4.5 Feature Selection

A feature in a maximum entropy model refers to a binary function. In NLP it typically expresses a co-occurrence relation between something in linguistic context and a particular prediction. For our lexical type prediction model, the features take the form of $f_i(t, c)$ where t is a potential prediction type and c is the context for the prediction (i.e., the word itself, its adjacent words, modifier/modified entity, ...). For example, the following feature holds value 1 for all the noun predictions when the candidate word that has suffix "–tion":

$$f_j(t, c) = \begin{cases} 1 & t = noun \quad \text{and} \quad word(c) = \text{``} * tion'' \\ 0 & \text{otherwise} \end{cases} \tag{3.15}$$

It should be noted that the number of features in a maximum entropy model is usually very large (typically hundreds of thousands). These features are created with feature templates. Good feature templates allow the creation of the most salient features and improve the model performance significantly.

In this dissertation, the selection of feature templates are based on the experiment results with the English Resource Grammar. Although the framework of the lexical type prediction is generally applicable to all languages, the detailed designs of feature templates are language specific. Therefore the discussion in the rest of this section should be considered as a specific application of the general model we have proposed so far.

Also, the feature selection is mainly concerned with the extractable features either from the input string, or the grammar related resource. Therefore the resulting features have minimum dependency on other language resources which are not always available for different languages (see Section 3.4.8 for more discussion).

Morphological Features

Morphological information is very helpful for determining the word category, even for lightly inflected languages like English. The study of part-of-speech tagging has shown that for English, the simple morphology handling with affix strings is very effective. Without going into a detailed morphological analysis for specific languages, I will also take the same approach. Furthermore, other morphological information (e.g., capitalization) is also taken into account.

Surface and Syntactic Context Features

The morphological features only concern the candidate word itself, not the context in which the word is used[5]. In order to take this into account, we need to gather extra information from the context. This includes the adjacent words, their corresponding atomic lexical types (assigned by the grammar, or *NULL* if it is not in the lexicon).

Preferably we should also include the features that can provide a better description of the syntactic context information, e.g., the head word, dependent relations, etc. However, since the sentence cannot be parsed due to the absence of the correct lexical entry, such information is not directly accessible. Since we intend to design the model as independent of extra language resources as possible, two simplified syntactic feature templates are used. When the sentence

[5]Note that the *context* here has a more strict sense to the definition of c in Section 3.4.4. c includes the candidate word, therefore embraces the morphological features, as well.

fails to parse, we retain the predominant left/right adjacent edges of the chart and use the corresponding grammar rule name as features.

For completeness, the feature templates discussed so far are listed in Table 3.3.

Morphological Features	
PREFIX$_i$ $1 \leq i \leq 4$	The i-character prefix of the word.
SUFFIX$_i$ $1 \leq i \leq 4$	The i-character suffix of the word.
CAPTAL	*true* if the word is capitalized, *false* otherwise.
DIGIT	*true* if the word contains digit (0-9), *false* otherwise.
HYPHEN	*true* if the word contains hyphen (-), *false* otherwise.
Syntactic Features	
LWORD$_i$ $i \in \{1, 2\}$	The i-th word to the left of current word.
RWORD$_i$ $i \in \{1, 2\}$	The i-th word to the right of current word.
LLEXTYPE$_i$ $i \in \{1, 2\}$	The lexical type of the i-th word to the left of current word.
RLEXTYPE$_i$ $i \in \{1, 2\}$	The lexical type of the i-th word to the right of current word.
LEDGE	The predominant left adjacent edge.
REDGE	The predominant right adjacent edge.

Table 3.3: Features groups

3.4.6 Incorporating Parse Disambiguation Results

As mentioned before, deep lexical types normally encode complicated constraints that only make sense when they work together with the grammar rules, and some subtle differences between lexical types do not show statistical significance in a corpus of limited size. Hence the feedback from later stages of deep processing is very important for predicting the lexical types for unknown words.

By incorporating the adjacent parsing edges as features, we somehow break the pipeline model of the processing. However, these features might help only when the candidate word is not the head of the phrase. Otherwise, the full parse disintegrates into small fragments, and the partial parsing results will not be able to provide discriminative information for the prediction model. An alternative way of breaking the pipeline model is to help the parser to generate full parses in the first place, and let the parsing result show which lexical entry is good.

In order to help the parser generate a full parse of the sentence, we feed the newly generated lexical entries directly into the parser. Instead of generating only one entry for each occurrence of the unknown word, we pass on top n most likely lexical entries. With these new entries, the sentence will receive one or more parses (assuming the sentence is grammatical and covered by the grammar). From the parsing results, a best parse is selected with the disambiguation model proposed by Toutanova et al. (2002), and the corresponding lexical entry is taken as the final result of lexical extension.

Within this processing model, the incorrect types will be ruled out if they are not compatible with the syntactic context. Also the infrequent readings of the unknown word will be dispreferred by the disambiguation model.

3.4.7 Experiments

To evaluate the effectiveness of our models, several experiments have been carried out for the *English Resource Grammar* (`ERG`) in combination with the `Redwoods` treebank (Oepen et al., 2002). We have used version *jun-04* of the grammar with the 5^{th} growths of the `Redwoods`.[6]

The *jun-04* version of the `ERG` defines in total 741 leaf lexical types, of which 709 are actually used (have at least one lexical entry) in its

[6] Several parts of the experiment have been repeated on later releases of the grammar and the treebank. The new results conform to the conclusion we derived from this version of the grammar and treebank. For completeness and to avoid confusion, only the results obtained from *jun-04* version of the `ERG` and the 5^{th} growth of the `Redwoods` are presented in this chapter without further notation.

lexicon with 12,347 entries. A large number of these lexical types are closed categories whose lexical entries should already be complete in the grammar. It is obvious that missing lexical entries, in most cases, should be in open categories. The major open categories are verbs, nouns, adjectives and adverbs. In the `ERG`, the number of leaf lexical types under these general categories are listed in Table 3.4.

General Cat.	Leaf Lex Types Num.
verb	261
noun	177
adjective	78
adverb	53

Table 3.4: Number of leaf lexical types under major open categories in `ERG`

Even for the open categories, the distribution of existing lexical entries over different lexical types varies significantly. Table 3.5 lists the top 10 lexical types with the maximum number of entries in the `ERG` lexicon.

Leaf Lexical Type	Num. of Entries
n_intr_le	1742
n_proper_le	1463
adj_intrans_le	1386
v_np_trans_le	732
n_ppof_le	728
adv_int_vp_le	390
*v_np*_trans_le*	342
n_mass_count_le	292
v_particle_np_le	242
n_mass_le	226

Table 3.5: Number of entries for top-10 leaf lexical types in `ERG`

The top 10 verbal types account for about 75% of the verbal entries.

For noun the figure is about 95% and 90% for adjectives. Presumably, this means that automated lexical extension for nouns will be easier. This is plausible because verbal lexical entries normally require more detailed subcategorization information.

For the evaluation, we used all the sentences with at least one preferred reading (reading generated by the grammar and proved to be correct by the human annotator) from the Redwoods Treebank. This gave us about 16.5K sentences and 122K tokens/words. All the experiments were done with 10-fold cross validation: the data were split into 10 partitions; each time 9 partitions were used as training data, and the remaining 1 for testing. For each fold, all the lexical entries which did not occur in the training set were assumed to be "missing" from the lexicon. The predictors predicted the atomic lexical type for each instance of a "missing" lexical entry (called testing instances). The corresponding treebanked entries were used as the gold standard. The *token accuracy* of the predictor on the test set is defined as the proportion of the correctly predicted atomic lexical types out of the total number of the testing instances.

For comparison, we have built a naïve baseline system that always assigns a majority type to each unknown according to the PoS tag. More specifically, we tagged the input sentence with a small Penn Treebank-like PoS tag-set. Then the PoS tag was mapped to a most popular lexical type for that PoS.[7] Table 3.6 lists part of the mappings.

PoS	Majority Lexical Type
noun	n_intr_le
verb	v_np_trans_le
adj.	adj_intrans_le
adv.	adv_int_vp_le

Table 3.6: Part of the PoS tags to lexical types mapping for ERG

Again for comparison, we have built another two simple prediction models with two popular general-purpose PoS taggers, TnT and

[7]This is similar to the built-in unknown word handling mechanism of the *PET* system.

MXPOST. TnT is a HMM-based trigram tagger while MXPOST is based on a maximum entropy model. We have trained the tagging models by using all the atomic lexical types as the tagset. The taggers tag the whole sentence, but only the output tags for the testing instances are taken in order to generate the lexical entries.

The result shows that the performance of the two taggers is very close. To evaluate how the size of a tagset might influence the prediction precision, another smaller tagset is used for comparison. According to the ERG lexicon, 350 leaf lexical types have no more than one entry, and 611 types have fewer than 10 entries. The top 30 types cover more than 75% of the entries. So in the smaller tagset, we use the top 20 most frequent open atomic lexical types and another 30 general PoS tags. All infrequent lexical types will be replaced by general PoS tags. For example, *"adj_wh_le"* and *"adj_poss_le"* will be replaced by *"adj"*; *"v_to_trans_le"* and *"v_pred_intrans_le"* will be replaced by *"v"*.

The tagger is trained with the smaller tagset likewise, but the top n possible tags are generated for each unknown word. The first atomic lexical type (non-general PoS) on the output list is chosen as the predictor's output. The experiment results will be reported below.

Figure 3.4 depicts the learning curves of the taggers with the different tagsets. The tagger with the smaller tagset slightly outperforms the tagger with the large tagset (approximately 1%). However, its learning curve is already getting flat. The tagger with the larger tagset is likely to match up if more training data is available.

The maximum entropy based model is tested both with and without using adjacent parsing edges as features. To incorporate the disambiguation results, our predictor generates 3 entries for each unknown word and stores them as temporary entries into the *LexDB*. The parse disambiguation model we used was proposed by Toutanova et al. (2002). It is essentially a maximum entropy based ranking model. Given an input sentence s with possible analyses $t_1 \ldots t_k$, the

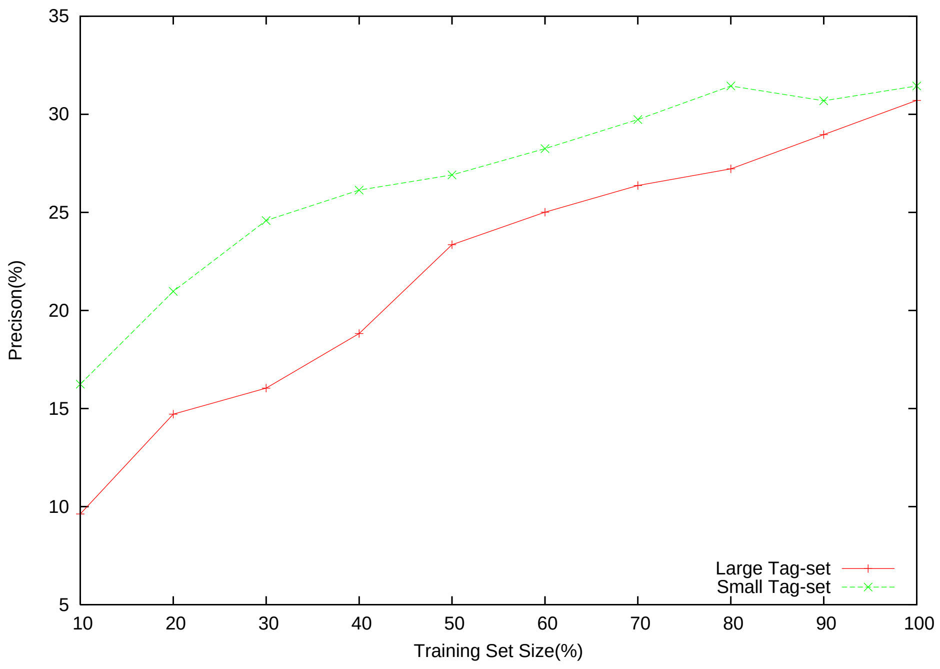


Figure 3.4: Learning curves of `TnT` with different tagsets

conditional probability for analysis t_i is given by:

$$P(t_i|s) = \frac{\exp \sum_{j=1}^{m} f_j(t_i)\lambda_j}{\sum_{i'=1}^{k} \exp \sum_{j=1}^{m} f_j(t_{i'})\lambda_j} \tag{3.16}$$

where $f_1 \ldots f_m$ are the features and $\lambda_1 \ldots \lambda_m$ are the corresponding parameters. When ranking parses, $\sum_{j=1}^{m} f_j(t_i)\lambda_j$ is the indicator of "goodness". Drawing on the discriminative nature of the ME models, various feature types can be incorporated into the model.

The *token accuracies* of the different prediction models are shown in Table 3.7.

The baseline model achieves token accuracy of around 30%. This means that the task of unknown word type prediction for deep grammars is non-trivial. The general-purpose `PoS` tagger-based models perform quite well, outperforming the baseline by 10%. As a confirmation to Elworthy (1995)'s claim, a huge tagset does not imply that tagging will be very difficult. Our ME-based model significantly outperforms the tagger-based models by another 10%. This is a strong

Model	Token Accuracy
Baseline	30.7%
TnT	40.4%
MXPOST	40.2%
ME(-LREDGE)	50.0%
ME(+LREDGE)	50.5%
ME(-LREDGE)+ disambi. result	61.3%

Table 3.7: Token accuracy of lexical type prediction models (+/-LREDGE means with or without adjacent parsing edge features)

indication of our model's advantages.

By incorporating simple syntactic information into the ME-based model, we get extra accuracy gain of less than 1%, which is not salient. The computational cost of obtaining such features is high, as well.

By incorporating the disambiguation results, the accuracy of the model is boosted by another 10%. The computational overhead is proportional to the number of candidate entries added for each unknown word. However, in most cases, introducing lexical entries with incorrect types will end up into parsing failure and can be efficiently detected by quick checking. In such cases the slowdown is acceptable.

In general, we have achieved up to 60% precision of unknown word type prediction for the ERG in these experiments. Given the complexity of the grammar and the huge number of possible lexical types, these results are satisfying. Also, in the real cases of grammar adaptation for new domains, a large portion of unknowns are proper names. This means that the precision might get even higher in real applications. A test with a small text collection with real unknown words [8] shows that the token accuracy can easily go above 80% with the

[8]We used a text set named *rondane* for training and *hike* for testing. Both of them are made available by the LOGON project. *rondane* contains 1424 sentences in formal written English about tourism in the Norwegian mountain area, with an average sentence length of 16 words; *hike* contains 320 sentences about outdoor hiking in

basic ME model without adjacent edge features.

It should also be mentioned that some of these experiments were also carried out for the Dutch `Alpino` Grammar (Bouma et al., 2001), and similar results were obtained. These results have been later reconfirmed independently by van de Cruys (2006), which roughly reimplemented a similar approach, with some fine tuning on feature selection. This shows that our method may be grammar and platform independent.

3.4.8 *In Vitro* vs. *In Vivo*

Baldwin (2005b) introduced the concept of *in vitro* and *in vivo* approaches to deep lexical acquisition. *In vitro* methods use a secondary lexical resource to model lexical similarity, whereas *in vivo* methods use some component of the target deep language resource for which there is an attempt to learn new lexical items to model lexical similarity.

According to the definition, our approach is a strict *in vivo* approach. We try to avoid using any feature in our ME-based model that cannot be derived without a secondary language resource. The deep grammar and the corresponding treebank are the only resources required for training/applying the model. And these treebanks are so-called dynamic treebanks, meaning that they can be (semi-)automatically updated when the grammar is updated, making them essentially a by-product of the grammar development.

The *in vivo* approach has advantages over *in vitro* models in that there is a minimum dependency on the availability of other language resources. Even if the secondary language resources are available, it is unclear whether they can provide the necessary and compatible lexical information for the deep language resource.

The potential advantage of the *in vitro* approach is that for major languages, there are usually more secondary language resources. Although the resources are not built specifically for deep processing,

Norway with an average sentence length of 14.3 words. Both contain a lot of unknowns like location names, transliterations, etc.

based on the same language, they can still provide lexical information, even though indirectly.

For purposes of comparison, another experiment was done to build a lexical acquisition model using the *in vitro* approach. More specifically, the new model extended the deep lexicon of the ERG using WordNet (version 2.0) as secondary language resource. Statistics shows that above 90% of the synsets in WordNet share at least one lexical type among all included words. By assuming that semantic similarity also entails syntactic similarity (Levin, 1993), for each candidate word/lexeme we first

1. use WordNet to construct the *"semantic neighbors"* (all synonyms, direct hyponyms, direct hypernyms) for the given word/lexeme;

2. take a majority vote across the atomic lexical types of the semantic neighbors.

This approach bootstrapped the deep lexicon from a different lexicon resource. This is essentially a similar model to that proposed by Baldwin (2005a). However, noticing the difference in the size of the deep lexicons (12,347 entries) and WordNet (152,059 unique strings/words and 115,424 synsets), this model performed poorly in reality because of insufficient votes from the "semantic neighbors". As an extension to this model, we built another improved model by:

1. extending the "semantic neighbors" (by including indirect hyponyms and hypernyms with distance of two); there the votes were weighted by their semantic distance to the candidate words (the smaller the distance, the larger the weight);

2. combining the majority type base-line system described earlier as fallback if the vote was less than a threshold.

The performance of the original model (*Baldwin05*) and the weighted voting model with threshold controlled fall-back (WVT) were evaluated with the type F-score of the resulting lexicon relating to the

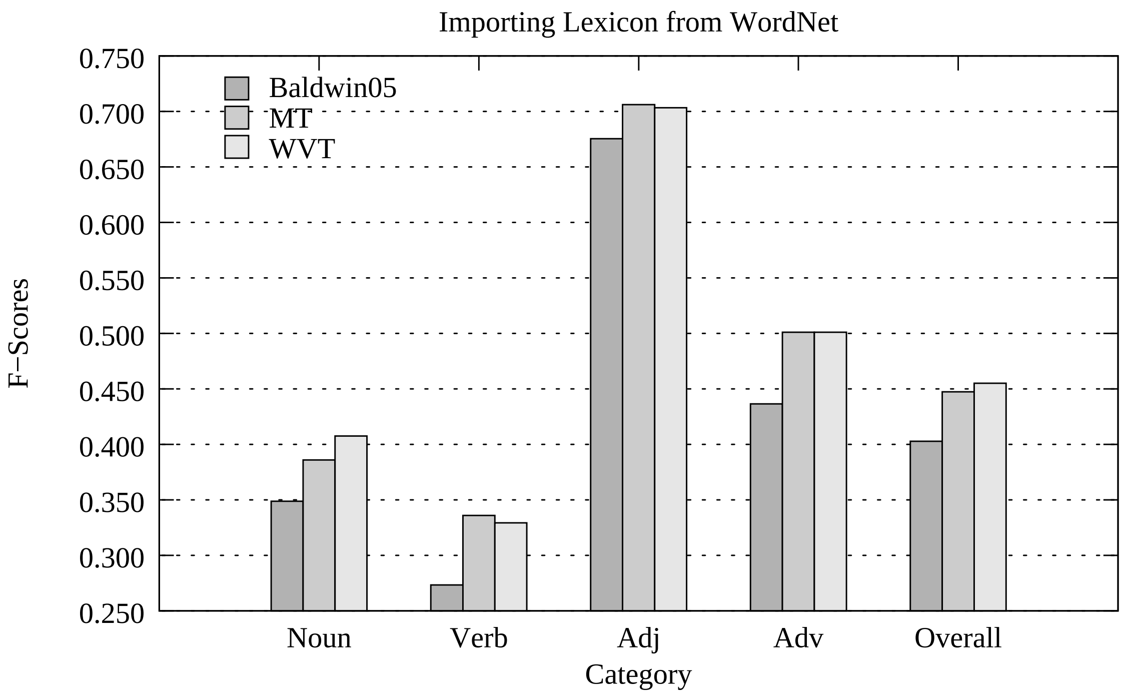


Figure 3.5: Performance of different models on `WordNet`-based deep lexicon bootstrapping

gold-standard lexicon. The results are shown in Figure 3.5, together with the majority type baseline (MT) described earlier.

It is clear that the *Baldwin05* model performs significantly worse than the baseline model MT on all the word categories. This shows that the different lexicon resources cannot be easily converted. In this specific case, `WordNet` is developed as a semantic lexical ontology, while the deep grammar lexicon is more focused on syntactic behaviors of different word categories.

With the weighted voting model of the threshold controlled fallback model WVT, the overall performance goes slightly above the majority type baseline. Admitted that with the combination of potentially large amounts of secondary language resources, the *in vitro* approach towards deep lexical acquisition can be promising, too, using extra language resources to enhance the robustness of deep processing is beyond the interest of this dissertation.

3.5 Summary

In this chapter, we discussed in detail approaches of automatically improving the lexical coverage of deep grammars. A relational view of the deep lexicon has been presented. Based on that, we have defined the atomic lexical types. It has been shown that manual lexical extension is neither efficient nor sufficient for the purpose of robust deep processing. Therefore we propose a two step deep lexical acquisition model. The first step detects the candidate words which are missing lexical entries, either via lexical gaps, or by error mining. In the second step, we propose a statistical approach of deep lexical acquisition as a classification task. Several models are proposed and evaluated with the English Resource Grammar and the `Redwoods` Treebank. In combination with parser output and disambiguation results, the token accuracy of the model goes above 60%. Furthermore, the relation of *in vivo* and *in vitro* approaches are also discussed.

The proposed methods improve the coverage of the grammar, and may also be used to enhance the robustness of deep processing in parsing tasks with noisy inputs. As mentioned in the previous chapter, lexicon coverage accounts for about 41% of the failed-to-parse cases in the `BNC` coverage test. Some of the research outcomes have been used in improving the English Resource Grammar and the method is to be generally applied to other grammars, platforms, or even formalisms.

In the next chapter, we will continue the discussion of improving lexical coverage and robustness for multiword expressions. The relation between the lexicon and the grammar performance will be discussed in Chapter 5.

4 Acquiring Multiword Expressions

Our expression and our words never coincide, which is why the animals don't understand us.

— Malcolm de Chazal *(1902 - 1981)*

In the previous chapter, we presented techniques to automatically acquire new lexical entries for the purpose of robust deep processing. Two main steps have been involved: i) the detection of words with missing lexical entries; ii) the classification and assignment of lexical types. So far we have assumed that each missing lexical entry corresponds to a single word, so that if the mapping from the lexeme to the lexical type is determined, the word will be handled correctly by the grammar. However, further complications do exist in real language usage. It has been observed in many languages that a certain group of words can be used together (with a certain degree of variation) in idiosyncratic ways. Despite the different nature of such phenomena, they are usually referred to as *Multiword Expressions* (MWEs), in general.

In the context of robust deep processing, multiword expressions also present a serious challenge. It has been estimated that the number of MWEs in a speaker's lexicon is of the same order of magnitude as the number of single words (Jackendoff, 1997). Also, the lack of method to systematically handle different MWEs makes them "a pain in the neck" for NLP (Sag et al., 2002).

In this chapter, we will show that the techniques developed in Chapter 3 can be adapted to handle MWEs. First, we will have a brief overview of multiword expressions in Section 4.1. Then the techniques to detect the boundaries of MWEs are introduced in Section 4.2. The MWE candidates are further validated with statistical measures described in Section 4.3, followed by two different approaches of acquiring new lexical entries for MWEs in Section 4.4. Further discussion is given in Section 4.5.

4.1 Multiword Expressions

The term Multiword Expressions has been used to describe expressions for which the syntactic or semantic properties of the whole expression cannot be derived from its parts (Sag et al., 2002), including a large number of related but distinct phenomena, such as phrasal verbs (e.g., *come along*), nominal compounds (e.g., *frying pan*), institutionalized phrases (e.g., *bread and butter*), and many others.

Jackendoff (1997) estimates the number of MWEs in a speaker's lexicon to be comparable to the number of single words. This is reflected in several existing grammars and lexical resources, where almost half of the entries are multiword expressions. However, due to their heterogeneous characteristics, MWEs present a tough challenge for both linguistic and computational work (Sag et al., 2002). For instance, some MWEs are fixed, and do not present internal variation, such as *ad hoc*, while others allow different degrees of internal variability and modification, such as *spill the beans* (*spill several/musical/mountains of beans*).

In terms of semantics, some MWEs are more opaque in their meaning (e.g., *to kick the bucket* as *to die*), while others have more transparent meanings that can be inferred from the words in the MWE (e.g., *eat up*, where the particle *up* adds a completive sense to *eat*).

Sag et al. (2002) discuss two main approaches commonly employed in NLP for treating MWEs: the *words-with-spaces* approach models an MWE as a single lexical entry, where the words of the MWE are separated by spaces. It can adequately capture fixed MWEs like *by and large*. A *compositional* approach treats MWEs by general and compositional methods of linguistic analysis, being able to capture more syntactically flexible MWEs, like *spill the beans*, which cannot be satisfactorily captured by a words-with-spaces approach, since it would require lexical entries to be added for all the possible variations of the MWE (e.g., *spill/spills/spilling some/these/... beans*). Therefore, to provide a unified account for the detection of these distinct but related phenomena is a real challenge for NLP systems.

4.2 Detecting MWE Candidates

Similar to the lexical acquisition for single words, the multiword expressions must be first detected before they can be properly handled in deep grammar. However discovering the *boundary* of MWEs is difficult due to their heterogeneous characteristics. For example, in the verb-particle constructions (e.g., *eat ... up*), the object occurs in between the verb and the particle, but it is not part of the MWE and allows a high level of variation.

Instead of discovering the exact MWEs in one step, we take an approximation step at the beginning in order to locate a consecutive sequence of words (also called an n-gram) which is part of the multiword expression that cannot be handled by the grammar.

Recall that in Section 3.3.3 we proposed to use an error mining technique to detect missing lexical entries. The intuition is that the parsing failure gives a good indication of the grammar errors. In Chapter 3, we have shown that the low parsability unigrams have strong correlation to the missing lexical entries for single words. However, the unigrams only account for about 1/3 of the low parsability n-grams (see Table 3.2). There is a large proportion of bigrams and trigrams, as well. With a closer look at these n-grams with larger n, we found that many of them, especially trigrams and above, are related to the lack of proper handling of MWEs in the grammar. Table 4.1 lists some of the low parsability n-grams with $n \geq 3$.

The list is potentially interesting, for many of them are directly related to multiword expressions. For example, *"by and large"*, *"points of view"* and *"foot the bill"* are good candidates of MWEs by themselves, which are missing from the lexicon of the ERG. However, there

N-gram	Count	Parsability
the burden of	49	0.000
by and large	37	0.000
face of it	34	0.000
frame of mind	23	0.000
points of view	20	0.000
hair and a	17	0.000
the to infinitive	15	0.000
let alone a	12	0.000
foot the bill	10	0.000
of alcohol and	8	0.000
a great many	44	0.083
glance up at	33	0.083
for and against	21	0.086

Table 4.1: Examples 3+grams from the results of error mining with ERG (*jan-06*) and BNC

are also non-MWE n-grams like *"of alcohol and"*, which correspond to other types of error of the grammar. Moreover, some of the n-grams are related to missing MWEs, but also include other words which should not be considered part of the MWE. For instance, *"let alone a"* in Table 4.1 is related to the MWE *"let alone"*, while the extra *"a"* in the n-gram indicates the collocation of the MWE with a nominal object.

The complication indicates that an extra validation step is required before the true MWE can be correctly captured. In this work, we call the resulting 3+grams (n-grams with $n \geq 3$) of error mining the MWE *raw candidates*. The validation methods presented in the following section are based on this candidate list.

4.3 MWE Candidate Validation

In order to discriminate the n-grams which represent good MWE candidates from erroneous ones, a series of validation steps are introduced.

First, with the MWE candidates obtained from the error mining results, some heuristic rules are used as filters in order to discard the obvious non-MWEs, following Bouma and Villada (2002). For example, those n-grams which contain one or more acronyms, names, or dates are not taken into further consideration.

For the remaining list of MWE candidates, we use further statistical measures for validation. Similar approaches have been reported by Pearce (2002) for collocations, and Zhang et al. (2006) for MWEs, in general. Different resources are used to gather the frequency of the n-grams and their variants. Besides large corpora like the BNC, many researchers have recently started using the web as an extremely large corpus, since, as pointed out by Grefenstette (1999), the Web is the largest data set available for NLP (Grefenstette, 1999; Keller et al., 2002; Kilgarriff and Grefenstette, 2003; Villavicencio, 2005). For instance, Grefenstette (1999) employs the Web to do example-based machine translation of compounds from French into English. The method he employs would suffer considerably from data sparseness, if it were to rely only on corpus data. So for compounds that are sparse in the BNC, he also obtains frequencies from the Web. The scale of the Web can help minimize the problem of data sparseness, which is especially acute for MWEs. Villavicencio (2005) uses the Web to find evidence to verify automatically generated verb-particle constructions.

Different statistical measures are available for the validation step. Here we only list some of the measures.

- *Mutual Information* (MI): is a useful quantity that measures the mutual dependence of multiple random variables. For a trigram with words $w_1w_2w_3$, Mutual Information is calculated as:

$$\text{MI} = \sum_{a,b,c} \frac{n(abc)}{N} \log_2 \left[\frac{n(abc)}{n_\emptyset(abc)} \right] \tag{4.1}$$

  where a corresponds either to the word w_1 or to $\neg w_1$ (all but the word w_1) and so on. $n(abc)$ is the number of trigrams abc in the corpus, $n_\emptyset(abc) = n(a)n(b)n(c)/N^2$ is the predicted number from the *null* hypothesis, $n(a)$ is the number of unigrams a, and N the number of words in the corpus.

- χ^2 test: is a standard statistical hypothesis test for independence between a group of variables. A χ^2 distribution with *degrees of freedom* k is defined in Equation 4.2. When each x_i corresponds to a word in the candidate n-gram, the test shows whether their occurrences are correlated or a random assembly.

$$\chi^2 = \frac{(x_1 - \mu_1)^2}{\sigma_1^2} + \frac{(x_2 - \mu_2)^2}{\sigma_2^2} + \cdots + \frac{(x_k - \mu_k)^2}{\sigma_k^2} \tag{4.2}$$

- *Permutation Probability* (PP): is used by Zhang et al. (2006) as a statistical measure for the ordering of the n-gram. For a given n-gram, there are at most $n!$ total permutations (or fewer, if there are duplicated words). Suppose there are m different n-grams as the result of permuting the original n-gram, we define the permutation probability of the i^{th} n-gram P_i as the ratio of the frequency count of the n-gram against the frequency count of the consecutive n words in any ordering.

- *Permutation Entropy* (PE): is also used by Zhang et al. (2006) and Villavicencio et al. (2007) as another statistical measure for the ordering of the n-gram. Based on the definition of permutation probability, the permutation entropy is defined as following:

$$PE = -\frac{1}{\log m} \sum_{k=1}^{m} P_i \log P_i \tag{4.3}$$

All of the above statistical measures can be obtained with either a static corpus (i.e., BNC) or the WWW. The technologies to access such huge amounts of data are far beyond the topic of this dissertation. In our work, we rely on the Web search engines to get a rough estimation of the frequencies of the given n-gram on web pages. Two different web search engines are used: Google and Yahoo.

We validated some of the low parsability n-grams from the error-mining results of the ERG (*jan-06*) using the permutation entropy and probability measures. Some example results are listed in Table 4.2.

MWE	HITS	PE	PP (%)
the burden of	36,600,000	0.366	79.4
and cost effective	34,400,000	0.372	70.7
the likes of	34,400,000	0.163	93.1
but also in	27,100,000	0.038	98.9
to bring together	25,700,000	0.086	96.6
points of view	24,500,000	0.017	99.6
and the more	23,700,000	0.512	61.5
with and without	23,100,000	0.074	97.4
can do for	22,300,000	0.003	99.9
taking into account the	22,100,000	0.009	99.6
but what about	21,000,000	0.045	98.7
the ultimate in	17,400,000	0.199	90.0
stand by and	1,350,000	0.399	65.5
discharged from hospital	553,000	0.001	99.9
shock of it	92,300	0.541	44.6
was woken by	91,400	0.001	99.9
telephone rang and	43,700	0.026	99.2
glanced across at	36,900	0.003	99.9
the citizens charter	22,900	0.070	97.9
from of government	706	0.345	0.1
the to infinitive	561	0.445	1.4

Table 4.2: Examples of the statistical validation results. *HITS* are the number of matching web pages reported by Google; PE are the permutation entropies; PP are the permutation probabilities

We see that the *HITS* is a good measure to eliminate the unlikely MWEs. High *PP* and low *PE* usually correspond to better MWE candidates. However, the two measures should be considered together to make a more reliable judgment. The basic hypothesis we make here is that the good MWE candidates are not random permutations of the words in the n-gram. A more detailed evaluation of different statistical measures in terms of discovering genuine MWEs have been reported by Villavicencio et al. (2007).

4.4 Acquiring Lexical Entries for MWEs

Recognition of the MWEs is just the first step. To handle them properly in deep processing requires more sophisticated techniques than those we have developed so far. Nevertheless, it is the key to the robustness problem considering the widespread use of MWEs. Also the techniques should be as automatic as possible. Given the heterogeneous nature of different MWEs, systematic treatment of the different phenomena is difficult, if possible at all. In this section, we will present two alternative approaches from the practical grammar engineering points of view, aiming at maximal robustness without much loss in grammar accuracy.

4.4.1 *Words-with-Spaces* Approach

The naïve, yet robust approach of handling MWEs is to treat the n-grams as single words. The only difference is that they may contain spaces. Obviously such simplification suffers from a serious limitation of its own (which we will discuss in details in Section 4.4.2). Nonetheless, this approach is widely in use in grammar engineering. For instance, from the 23K lexical entries in the ERG (jan-06) lexicon, more than 1.7K (7.5%) entries correspond to words with more than one tokens. In fact, for some types of MWEs which allow minimal variation (i.e., fixed expressions like *ad hoc*), the *words-with-spaces* approach suffices. For semi-fixed expressions, like compound nominals and proper names, appropriate morphological handling is necessary. But for syntactically flexible MWEs, like verb-particle constructions, this approach is less adequate. Even though less accurate, the *words-with-spaces* approach can in most cases encapsulate the difficult MWEs into small lexical units, and avoid the in-depth analysis within the n-gram. The resulting analysis will suffer a decrease in accuracy, but with much better coverage in the meantime.

With minimum modification to our lexical type prediction model, the MWE entries can be acquired from large corpora automatically. In Zhang et al. (2006), we reported that with the ERG (*jan-06*) error-mining results, 311 n-grams were validated as good MWE candidates. Then 6,246 sentences were extracted from the fragment of the BNC corpus (cf., Section 3.3.3) which contained at least one of the n-grams. The lexical type prediction model trained on the Redwoods treebank was used to predict the lexical type for each occurrence of the n-grams. However, not all the predictions were accepted for generating new lexical entries. Only those predictions which occurred over a minimum threshold (5 in this case) were accepted.

For the ERG (*jan-06*), we obtained 373 new *"words-with-spaces"* lexical entries. The grammar coverage on the 6,246 sentences before and after adding these entries presented a significant difference. Details are given in Table 4.3.

	item #	+entry #	parsed #	analysis ϕ	coverage %
-MWE	6246	–	268	209.76	4.3%
+MWE	6246	373	1168	137.14	18.7%

Table 4.3: ERG coverage on "difficult" sentences before and after adding "word-with-spaces" MWE lexical entries

4.4.2 Compositional Approach

Despite the significant improvement in coverage, the accuracy of the grammar was not investigated for the *"words-with-spaces"* approach in Zhang et al. (2006).

As Sag et al. (2002) pointed out, such an approach has several major limitations. First, the approach suffers from a *flexibility problem*. For example, Sag et al. (2002) pointed out that a parser which lacks sufficient knowledge of verb-particle constructions might correctly assign *look up the tower* two interpretations (*"glance up at the* *tower"* vs. *"consult a reference book about the tower"*), but fail to treat the subtly different *look the tower up* as unambiguous (*"consult a reference book ... "* interpretation only). For highly variable `MWE`s with such or other kinds of flexibility, the linguistic precision will not be fully captured in the *words-with-spaces* approach. Moreover, this simple approach also suffers from a *lexical proliferation problem*. For example, light verb constructions often come in families, e.g., *take a walk*, *take a hike*, *take a trip*, *take a flight*, ... Listing each such expression results in considerable loss of generality and lack of prediction.

A closer look at the `MWE`s not properly handled by the grammar reveals that only a small proportion of them can be handled appropriately by the "words-with-spaces" approach of Zhang et al. (2006). Simply adding new lexical entries for all `MWE`s can be a workaround for enhancing the parser coverage, but the quality of the parser output is clearly linguistically less adequate.

On the other hand, we also find that a large proportion of `MWE`s that cannot be correctly handled by the grammar can be covered properly in a compositional way by adding one lexical entry for the head (governing) word of the `MWE`. For example, the expression *foot the bill* will be correctly handled with a standard head-complement rule, if there is a transitive verb reading for the word *foot* in the lexicon. Some other examples are: *to **put** forward*, *the **good** of*, *in **combination** with*, ..., where lexical extension to the words in ***bold*** will allow the grammar to cover the `MWE`s. In this section, we focus on the constructional approach for the acquisition of new lexical entries for the head words of the `MWE`s.[1]

It is arguable that such an approach may lead to some potential grammar overgeneration, as there is no selectional restriction expressed in the new lexical entry. However, as far as the parsing task is concerned, such overgeneration is not likely to reduce the accuracy of the grammar significantly as we will show through a thorough

[1]The combination of the "words-with-spaces" approach with the constructional approach we propose here is an interesting topic that we want to investigate in future research.

evaluation.

With a similar setting to the one described in Section 4.4.1, we did two more acquisition experiments with two lists of candidate MWEs. The first list (L-I henceforth) contains 200 randomly selected MWEs from the high permutation entropy, high WWW *HITS* n-grams. The second list (L-II henceforth) is selected with a more thorough validation step by combining three statistical measures: the mutual information (MI), χ^2 test, and permutation probability (PP). The candidate MWEs are ranked according to all these measures, and the top 30 MWEs with the highest average rankings are selected into the second list. The difference with Section 4.4.1 is that these evaluations are done on a slightly more recent version of the ERG (*jul-06*)[2].

With both lists of MWEs, the following steps are taken to find the head word with heuristics:

- the n-grams are PoS tagged with an automatic tagger;
- finite verbs in the n-grams are extracted as head words;
- nouns are also extracted if there is no verb in the n-gram.

Occasionally, tagger errors might introduce wrong head words. However, the lexical type predictor of Zhang and Kordoni (2006) that we used in our experiments did not generate interesting new entries for them in the subsequent steps. As a result, we obtained 52 head words for L-I and 20 for L-II.

With the two lists of MWEs, we extracted two sub-corpora from the BNC, with each sentence containing at least one of the MWEs in the corresponding lists. The sub-corpora contain 2,463 sentences and 674 sentences, respectively. The lexical acquisition technique described by Zhang and Kordoni (2006) was used with these sub-corpora in order to acquire new lexical entries for the head words. The lexical

[2]The major difference between the ERG version *jan-06* and *jul-06* is the size of the lexicon. *jul-06* shows much better overall coverage on the BNC, largely due to the semi-automatic lexical extension. But no significant change in the grammar accuracy is observed.

acquisition model was trained with the `Redwoods` treebank, following the techniques presented in Chapter 3.

The lexical prediction model predicted the most plausible lexical type in that context for each occurrence of the head words. Only those predictions that occurred 5 times or more were taken into consideration for the generation of the new lexical entries. As a result, we obtained 50 new lexical entries for L-I and 21 for L-II.

These new lexical entries were later merged into the `ERG` lexicon. To evaluate the grammar performance with and without these new lexical entries, we

1. parsed the sub-corpora with and without new lexical entries and compared the grammar coverage;

2. inspected the parser output manually and evaluated the grammar accuracy.

In parsing the sub-corpora, we used the `PET` parser (Callmeier, 2001). For the manual evaluation of the parser output, we used the treebanking tools of the `[incr tsdb()]` system (Oepen, 2001).

Table 4.4 shows that for L-I the grammar coverage improved significantly (from 4.3% to 16.7%) with the acquired lexical entries for the head words of the `MWE`s. With a more carefully validated list L-II, the coverage improvement is even more noticeable (over 15%, see Table 4.5). These improvements in coverage are largely comparable to the result we observed in the *"words-with-space"* approach (from 4.3% to 18.7%). Also, we discovered that with a more carefully filtered list of candidates, the average analysis number drops when new lexical entries are added, somehow indicating that those new entries do not lead to terribly more readings with higher lexical ambiguity.

Comparing the numbers of the new lexical entries added, we noticed that the compositional approach achieved comparable coverage improvement with fewer new lexical entries. This suggests that the lexical entries acquired in our experiment are of much higher linguistic generality.

To evaluate the grammar accuracy, we manually checked some of the parser outputs from each sub-corpus after the lexical extension.

	item #	+entry #	parsed #	analysis ϕ	coverage %
-MWE	2463	–	107	127.88	4.3%
+MWE	2463	52	412	178.33	16.7%

Table 4.4: ERG coverage with and without lexical acquisition for the head words of L-I MWEs (compositional)

	item #	+entry #	parsed #	analysis ϕ	coverage %
-MWE	674	–	48	335.08	7.1%
+MWE	674	21	153	285.01	22.7%

Table 4.5: ERG coverage with and without lexical acquisition for the head words of L-II MWEs (compositional)

A sentence was marked as "accepted", if one of the analyses was correct, otherwise it was marked as "rejected". The results are listed in Table 4.6.

lists	sentence #	accepted #	%
L-I	100	81	81.0%
L-II	153	124	81.0%

Table 4.6: ERG accuracy after lexical acquisition for the head words of MWEs

Baldwin et al. (2004) reported earlier that, for BNC data, about 83% of the sentences covered by the ERG have a correct parse. In our evaluation, we observed very similar accuracies. We also found that the disambiguation model as described by Toutanova et al. (2002) performed reasonably well, and the best analysis is ranked among top-5 for 66% of the cases, and top-10 for 75%.

All of these results indicate that our approach for lexical acquisition of head words of MWEs achieves a significant improvement in grammar coverage without damaging the grammar accuracy. Optionally, the grammar developers can check the validity of the lexical entries before they are added into the lexicon. This semi-automatic procedure can largely reduce the manual work of grammar writers.

4.5 Discussion

Multiword expressions are a class of heterogeneous language phenomena which are prevalent in use, but lack systematic treatment. In this dissertation, we started from a grammar engineering perspective, aiming at the maximal robustness of deep processing. Comparing to the other studies of multiword expressions, we see that our approach has the following advantages:

- It is highly automatic. With various statistical measures over large corpora, as well as learning mechanisms, the entire process requires minimal human intervention.

- It is not phenomenon specific. Our methods provide a practical solution to effectively handle different types of MWEs in a consistent way. Although fine-grained classification of different MWE phenomena can be helpful in improving the performance on specific types of MWEs, our approach is much more general.

- It is not language specific. Our approach does not rely on any language specific presumption. Therefore, the methods can be easily adapted to work for different languages and grammars.

4.6 Summary

In this chapter, we have described techniques to automatically discover and handle multiword expressions from the grammar engineering perspective for maximal robustness. The error mining results provide us with MWE candidates in the form of low parsability n-grams. The statistical analyses on large corpora (BNC and WWW) help discriminate good candidates from noise. With good MWE candidates, we take either the *"words-with-spaces"* or the compositional approach in order to generate new lexical entries. Through a series of experiments we have shown that both methods are able to improve the grammar coverage significantly, while the compositional approach is also able to maintain the grammar accuracy.

5 Evaluation Metrics for Deep Lexical Acquisition

Our deepest fear is not that we are inadequate. Our deepest fear is that we are powerful beyond measure. It is our Light, not our Darkness, that most frightens us.

— Marianne Williamson, *"Return to Love, 1992"*

In the previous two chapters, we have discussed techniques for improving the lexicon coverage and robustness. The ultimate goal is to help boost deep grammar coverage and robustness in deep processing. The evaluation presented in the previous chapters shows that the techniques developed so far deliver promising performance on the module level. However, it is still unclear how the lexicon precision and recall may be related to the grammar performance. In this chapter, a series of experiments are reported in an attempt to unveil the correlation between precision and recall, on the one hand, and deep grammar performance, on the other.

5.1 The *"Goodness"* of Deep Lexical Acquisition

As mentioned earlier in Section 3.3.1, there are two types of lexical errors: i) a lexical entry is missing from the lexicon; ii) an erroneous entry enters the lexicon. The first type of error normally leads to undergeneration of the grammar, while the latter usually causes overgeneration.

Without considering the interaction with the grammar performance, the quality of the lexicon can be measured with standard precision and recall metrics. The precision is usually defined as the proportion of the correct entries among all the entries in the lexicon. The recall is defined as the proportion of correct entries in the lexicon among all the correct entries for the language (which are not necessarily in the lexicon).

However, in evaluating a lexicon, such definitions are difficult to follow. On the one hand, the *"correctness"* of the existing lexical entries is difficult to judge. Also, it heavily depends on the grammatical analysis. Therefore the precision of the lexicon is, at best, a subjective measurement. On the other hand, the measurement of recall is even more difficult, as there is no direct way to properly estimate how many lexical entries are missing from the lexicon.

Therefore, precision and recall are usually defined relative to a gold standard lexicon (denoted by L). The gold standard lexicon is usually built manually, therefore contains very few erroneous entries. Also, it must cover most of the lexical usage for a specific corpus. Therefore the recall of the lexicon for this specific corpus is also high. For a given lexicon L' as a set of lexical entries, it can be partitioned into two subsets:

$$L' = G \cup E \quad (G \cap E = \phi) \tag{5.1}$$

where G is the set of the entries that also belong to the gold standard lexicon L:

$$G = \{g \in L' | g \in L\} \tag{5.2}$$

and E is the set of the entries that do not occur in L, and are considered to be errors:

$$E = \{e \in L' | e \notin L\} \tag{5.3}$$

.

The precision P and recall R of the lexicon L' relative to gold standard lexicon L are defined as:

$$P = \frac{|G|}{|L'|} \tag{5.4}$$

$$R = \frac{|G|}{|L|} \tag{5.5}$$

.

The relative precision and recall more or less reflect the similarity of the lexicon to the gold standard. However, their limitations are very obvious.

First, the availability of the so-called gold standard lexicon is questionable. The starting motivation of deep lexical acquisition is to help build the lexicon. The imperfection of the "gold" lexicon renders the similarity based evaluation less reliable. Any entry in the "gold" lexicon is taken for granted as correct. Any entry that is not in the "gold" lexicon is considered to be an erroneous entry.

One way to balance this bias is to restrict the evaluation on a sub-language, bounded with a corpus. A high quality "gold" sub-lexicon can be extracted from an existing larger lexicon. This sub-lexicon satisfies the following two conditions:

- All the lexical usage in the corpus is included in the sub-lexicon;
- All the entries in the sub-lexicon correspond to at least one usage (instance) in the corpus.

If the deep lexical acquisition models are built and evaluated on this corpus using the relative precision and recall to the "gold" sub-lexicon, and if the corpus is well balanced and representative of the entire language, the evaluation results can be indicative of the true precision and recall of the model.

If the above question is still amendable, the following question is even more crucial. Suppose the relative precision and recall of the lexicon reflects its true precision and recall, it is still largely unclear how these figures are related to the grammar performance. Simple conjecture will be that the erroneous lexical entries lead to overgeneration of the grammar, and therefore making the parser outcome less precise. On the other hand, missing lexical entries lead to undergeneration of the grammar, hence hurting the coverage of the parser. Therefore, the precision of the lexicon should correspond to the accuracy of the grammar (in parsing tasks), while lexicon recall is related to the coverage. Unfortunately, in practice the interaction between the lexicon and the grammar is much more subtle. For example, besides the overgenerating effect, erroneous lexical entries might also cause parser failure (e.g., triggering recursive unary rules, exhausting parser memory by higher lexical ambiguities, etc.). On the other hand, the undergenerating effect of different missing lexical entries varies, depending on their frequency in the text corpus. A more frequent missing entry has a much larger effect on the grammar coverage. This correlation is not directly reflected by the recall of the lexicon, either.

In summary, the precision and recall based evaluations of the lexicon are neither reliable by themselves, nor indicative about the grammar performance.

Then what about the token accuracy based evaluation used in Chapter 3? By working with a lexically annotated corpus, the token accuracy takes the lexical entry frequency into account. Also, by assuming that frequent words are already in the lexicon, the token accuracy is only measured for infrequent words, hence the risk of running into high lexical ambiguity and abnormal grammar behavior gets reduced. However, there is still no direct correlation to the grammar performance.

Despite all the difficulties, a good evaluation metric is crucial for the development of deep lexical acquisition. Different measurement will eventually lead to different design. Eventually we hope to maximally improve the overall average performance of the grammar. Therefore, a thorough investigation of how the lexicon performance correlates to the grammar performance should be done.

5.2 Experimental Setup

In this chapter, we are going to unveil the correlation between the lexicon performance and grammar performance via a set of experiments with large scale deep grammars, together with corresponding treebanks. The basic idea is to randomly introduce lexical errors to a “gold” sub-lexicon in order to simulate the results of deep lexical acquisition at different (relative) precision and recall levels. The resulting lexicons are then used with the grammar for processing the treebank again. More specifically, in this chapter we focus on the grammar performance in parsing tasks[1]. Therefore the grammar performance is evaluated by parser coverage and accuracy. In this section, the experiment setup is described in detail.

5.2.1 Resources

Grammars

For this experiment, two large scale HPSGs are used: the English Resource Grammar (ERG; Flickinger (2000)) and the JaCY (Siegel and Bender, 2002) for Japanese. Both of them aim at broad coverage of the respective languages with high accuracy. They are developed using the DELPH-IN infrastructure, and can be used with the same processing tools (i.e., LKB, PET, [incr tsdb()]). Besides the various similarities of the two grammars, they also represent interesting differences. The two languages belong to different language families. English has more strict word order, while Japanese allows for more freedom in the word order. The ERG we used in this experiment is the *jan-06* version of the grammar, which contains about 23K lexicon entries and more than 800 leaf lexical types. The *November 2005* version of JaCY is used, which contains a 48K lexicon and more than 300 leaf lexical types.

Treebanks

To test the grammar coverage and accuracy, we use two treebanks: Redwoods (Oepen et al., 2002) for English and Hinoki (Bond et al., 2004) for Japanese. These treebanks are so-called *dynamic treebanks*, which means that they can be (semi-) automatically updated when the grammar is updated. This feature is especially useful when we

[1]In generation, we tend to have a semantic representation as input, which is linked to pre-existing lexical entries. Hence, lexical acquisition has no direct impact on generation.

want to evaluate the grammar performance with different lexicon configurations.

With conventional treebanks, our experiment is difficult (if not impossible) to perform as the static trees in the treebank cannot be easily synchronized to the evolution of the grammar, meaning that we cannot regenerate gold-standard parse trees relative to a given lexicon (especially when for reduced recall, there is no guarantee that we will be able to produce all of the parses in the 100% recall gold-standard). As a result, it is extremely difficult to faithfully update the statistical models.

The Redwoods treebank we use is the *6th growth*, which is synchronized with the *jan-06* version of the ERG. It contains about 41K test items in total.

The Hinoki treebank we use is updated for the November 2005 version of the JaCY grammar. The *"Rei"* section we use in our experiment contains 45K test items in total.

5.2.2 Lexicon Generation

To simulate deep lexical acquisition results at various levels of precision and recall, a random lexicon generator is used. In order to generate a new lexicon with specific precision and recall, the generator randomly retains a portion of the gold-standard lexicon, and generates a pre-determined number of erroneous lexical entries.

More specifically, for each grammar we first extract a subset of the lexical entries from the lexicon, each of which has at least one occurrence in the treebank. These subsets of lexical entries are considered to be the gold-standard sub-lexicons (7,156 entries for the ERG, 27,308 entries for JaCY)[2]. The simulated lexicons will be generated at different precision and recall levels relative to these gold-standard sub-lexicons.

[2]The size of the gold-standard sub-lexicons differ significantly mainly due to the different vocabulary in use in different treebanks. The Hinoki *"Rei"* section is based on the dictionary example sentences, while the Redwoods is based on relatively simple dialogs (i.e., travel planning, appointment arrangement, email correspondence, etc.).

Given the gold-standard lexicon L, the target precision P and recall R, a new lexicon L' is created, which is composed of two disjoint subsets: the retained part of the gold-standard lexicon G, and the erroneous entries E. According to the definitions of precision and recall:

$$P = \frac{|G|}{|L'|} \tag{5.6}$$

$$R = \frac{|G|}{|L|} \tag{5.7}$$

and the fact that:

$$|L'| = |G| + |E| \tag{5.8}$$

we get:

$$|G| = |L| \cdot R \tag{5.9}$$

$$|E| = |L| \cdot R \cdot (\frac{1}{P} - 1) \tag{5.10}$$

To retain a specific number of entries from the gold-standard lexicon, we randomly select $|G|$ entries based on the combined probabilistic distribution of the corresponding lexeme and lexical types.[3] We obtain the probabilistic distribution of lexemes from large corpora (`BNC` for English and Mainichi Shimbun [1991-2000] for Japanese), and the distribution of lexical types from the corresponding treebanks. For each lexical entry $e(l,t)$ in the gold-standard lexicon with lexeme l and lexical type t, the combined probability is:

$$p(e(l,t)) = \frac{C_L(l) \cdot C_T(t)}{\sum_{e'(l',t') \in L} C_L(l') \cdot C_T(t')} \tag{5.11}$$

where $C_L(l)$ and $C_T(t)$ are the frequency counts of lexeme l and lexical type t in corresponding corpora and treebanks.

The erroneous entries are generated in the same way among all possible combinations of lexemes and lexical types. The difference is that only open category types and less frequent lexemes are used for

[3]For simplicity, we assume mutual independence of the lexemes and lexical types.

generating new entries (e.g., we would not expect to learn a new lexical item for the lexeme *the* or the lexical type *d_-_the_le* in English). This also avoids abnormal grammar behavior that would be caused by having erroneous closed category entries.

In our experiment, we consider lexical types with more than a predefined number of lexical entries (20 for the `ERG`, 50 for `JaCY`) in the gold-standard lexicon to be open-class lexical types; the upper-bound threshold on token frequency is set to 1000 for English and 537 for Japanese, i.e., lexemes which occur more frequently than this are excluded from lexical acquisition under the assumption that the grammar developers will have attained full coverage of lexical items for them.

For each grammar, we then generate 9 different lexicons at varying precision and recall levels, namely 60%, 80%, and 100%.

P \ R	0.6			0.8			1.0		
	C	E	A	C	E	A	C	E	A
0.6	4294	2862	7156	5725	3817	9542	7156	4771	11927
0.8	4294	1073	5367	5725	1431	7156	7156	1789	8945
1.0	4294	0	4294	5725	0	5725	7156	0	7156

Table 5.1: Different lexicon configurations for the `ERG` with the number of correct (C), erroneous (E) and combined (A) entries at each level of precision (P) and recall (R)

P \ R	0.6			0.8			1.0		
	C	E	A	C	E	A	C	E	A
0.6	16385	10923	27308	21846	14564	36410	27308	18205	45513
0.8	16385	4096	20481	21846	5462	27308	27308	6827	34135
1.0	16385	0	16385	21846	0	21846	27308	0	27308

Table 5.2: Different lexicon configurations for the `JaCY` with the number of correct (C), erroneous (E) and combined (A) entries at each level of precision (P) and recall (R)

5.2.3 Parser Coverage

Coverage is an important grammar performance measurement, and indicates the proportion of inputs for which a correct analysis has been obtained. In parsing tasks, the parser coverage is the proportion of inputs for which a correct parse is reported, judged relative to the gold-standard parse data in the treebanks.

In this experiment, we adopt a weak definition of coverage as "obtaining at least one spanning tree". The reason for this is that we want to obtain an estimate for novel data (for which we do not have gold-standard parse data) from the relative number of strings for which we can expect to be able to produce at least one spanning parse. This weak definition of coverage actually provides an upper bound estimate of coverage in the strict sense, and spares us the effort to manually evaluate the correctness of the parses. Past evaluations (e.g., Baldwin et al. (2004)) have shown that the grammars we are dealing with are relatively precise. Based on this, we claim that our results for parse coverage provide a reasonable estimate indication of parse coverage in the strict sense of the word.

In principle, coverage will only decrease when the lexicon recall goes down, as adding erroneous entries should not invalidate the existing analyses. However, in practice, the introduction of erroneous entries increases lexical ambiguity dramatically, readily causing the parser to run out of memory. Moreover, some grammars use recursive unary rules which are triggered by specific lexical types. Here again, erroneous lexical entries can lead to "fail to parse" errors.

Given this, we run the coverage tests for the two grammars over the corresponding treebanks: `Redwoods` and `Hinoki`. The maximum number of passive edges is set to 10K for the parser. We used `[incr tsdb()]` (Oepen, 2001) to handle the different lexicon configurations and data sets, and `PET` (Callmeier, 2000) for parsing.

5.2.4 Parser Accuracy

Another important measurement of grammar performance is accuracy. Depending on the ambiguity level of the input, the number of analyses can vary from zero to tens of thousands. Not all the analyses are interesting for the applications, suggesting the need for some means of selecting the most probable analysis among them.

In the PET parser, this is done with a parse disambiguation model proposed by (Toutanova et al., 2002). Please refer to Section 3.4.7 for a brief description of the model. In combination with the dynamic treebanks where the analyses are (semi-)automatically disambiguated, the models can be easily re-trained when the grammar is modified.

In this experiment, the parser accuracy indicates the proportion of inputs for which we are able to accurately select the correct parse.

For each lexicon configuration, after the coverage test, we do an automatic treebank update, and train/evaluate the ME-based parse disambiguation models with 5-fold cross validation. Since we are only interested in the difference between different lexicon configurations, we use the simple PCFG-S model of Toutanova et al. (2002), which incorporates PCFG-style local tree branching features from the derivation tree of the parse. The accuracy of the disambiguation model is calculated by a top analysis exact matching (i.e., a ranking is only considered correct if the top ranked analysis matches the gold standard preferred reading in the treebank).

All the Hinoki *Rei* noun sections (about 25K items) were used in the accuracy evaluation for JaCY. However, due to technical limitations, only the *JH* sections (about 6K items) of the Redwoods Treebank were used for training and testing the disambiguation models for the ERG.

5.3 Experiment Results

The experiment has consumed a considerable amount of computational resources. For each lexicon configuration of a given grammar, we need to i) process (parse) all the items in the treebank, ii) compare the resulting trees with the gold-standard trees and update the treebank, and iii) retrain the disambiguation models over 5 folds of cross validation. Given the two grammars with 9 configurations each, the entire experiment has taken over 1 CPU month and about 120GB of disk space.

The coverage results are shown in Table 5.3 and Table 5.4 for JaCY and the ERG, respectively. It is worth noting that even with the lexicons at 100% precision and recall levels, there is no guarantee of 100% coverage on the treebank data. As the contents of the Redwoods and Hinoki treebanks have been determined independently of the respective grammars, rather than the grammars being induced from the treebanks, for instance, they both still contain significant numbers of strings for which the grammar cannot produce a correct analysis. As expected, we see a significant increase in grammar coverage when the lexicon recall goes up. This increase is more significant for the ERG than JaCY, mainly because the JaCY lexicon is about twice as large as the ERG lexicon; thus, due to the Zipfian distribution of the lexical entries, the most frequent entries are still in the lexicons even with low recall.

When the lexicon recall is fixed, the grammar coverage does not change significantly at different levels of lexicon precision. The coverage of low precision lexicons is usually slightly higher, as erroneous entries have a chance of accidentally generating a spanning parse. Recall that we are not evaluating the correctness of such parses at this stage.

It is clear that the increase in lexicon recall boosts the grammar coverage, as we would expect. The precision of the lexicon does not have a large influence on coverage. This result confirms that with DLA (where we hope to enhance lexical coverage relative to a given corpus/domain), the coverage of the grammar can be enhanced significantly.

P \ R	0.6	0.8	1.0
0.6	44.56%	66.88%	75.51%
0.8	42.18%	65.82%	75.86%
1.0	40.45%	66.19%	76.15%

Table 5.3: Parser coverage of JaCY with different lexicons

P \ R	0.6	0.8	1.0
0.6	27.86%	39.17%	79.66%
0.8	27.06%	37.42%	79.57%
1.0	26.34%	37.18%	79.33%

Table 5.4: Parser coverage of the ERG with different lexicons

The accuracy results are obtained with 5-fold cross validation, as shown in Table 5.5 and Table 5.6 for JaCY and the ERG, respectively. When the lexicon recall goes up, we observe a small but steady decrease in the accuracy of the disambiguation models, for both JaCY and ERG. This is generally a side effect of change in coverage: as the grammar coverage goes up, the parse trees become more diverse, and are hence harder to discriminate.

P-R	#ptree	fold1	fold2	fold3	fold4	fold5	Avg.
060-060	13269	63.70%	62.72%	61.56%	63.77%	61.52%	62.65%
060-080	19800	60.11%	61.05%	61.00%	58.95%	61.73%	60.57%
060-100	22361	59.75%	61.05%	59.33%	58.46%	59.45%	59.61%
080-060	14701	62.50%	62.93%	62.85%	63.97%	64.13%	63.27%
080-080	23184	61.10%	61.29%	61.53%	61.16%	59.76%	60.97%
080-100	27111	58.78%	59.63%	60.48%	60.80%	60.52%	60.04%
100-060	15696	63.24%	64.35%	64.76%	62.98%	64.22%	63.91%
100-080	26859	59.76%	61.57%	62.76%	61.50%	61.76%	61.47%
100-100	31870	61.27%	61.48%	59.29%	59.96%	60.42%	60.48%

Table 5.5: Accuracy of the disambiguation models for JaCY with different lexicons

When the recall is fixed and the precision of the lexicon goes up, we observe a very small accuracy gain for JaCY (around 0.5% for each 20% increase in precision). This shows that the grammar accuracy gain is limited as the precision of the lexicon increases, i.e., that the disambiguation model is remarkably robust to the effects of noise.

It should be noted that for the ERG we failed to observe any accuracy gain at all with a more precise lexicon. This is partly due to the limited size of the updated treebanks. For the lexicon configuration 060 – 060, we obtained only 737 preferred readings/trees to train/test the disambiguation model over. The 5-fold cross validation

P-R	#ptree	fold1	fold2	fold3	fold4	fold5	Avg.
060-060	737	69.23%	79.80%	65.43%	69.80%	71.29%	71.11%
060-080	1093	58.94%	63.95%	62.16%	70.81%	63.83%	63.94%
060-100	3416	61.24%	59.38%	63.18%	61.43%	59.40%	60.92%
080-060	742	71.26%	69.14%	72.63%	67.78%	69.57%	70.07%
080-080	1282	57.31%	63.74%	59.47%	59.65%	68.86%	61.81%
080-100	3842	58.14%	57.28%	62.31%	58.67%	58.85%	59.05%
100-060	778	68.82%	77.66%	65.85%	63.04%	73.40%	69.76%
100-080	1440	54.00%	61.08%	63.68%	63.46%	60.73%	60.59%
100-100	4689	54.25%	57.79%	59.11%	57.57%	56.41%	57.03%

Table 5.6: Accuracy of the disambiguation models for `ERG` with different lexicons

results vary within a margin of 10%, which means that the models are still not converging. However, the result does confirm that there is no significant gain in grammar accuracy with a higher precision lexicon.

Finally, we combine the coverage and accuracy scores into a single F-measure ($\beta = 1$) value. The results are shown in Table 5.7, Table 5.8 and Figure 5.1. Again we see that the difference in lexicon recall has a more significant impact on the overall grammar performance than precision.

P \ R	0.6	0.8	1.0
0.6	52.08%	63.57%	66.62%
0.8	50.62%	63.30%	67.03%
1.0	49.54%	63.74%	67.42%

Table 5.7: Grammar performance (F-score) of `JaCY` with different lexicons

P \ R	0.6	0.8	1.0
0.6	40.03%	48.58%	69.04%
0.8	39.04%	46.62%	67.79%
1.0	38.24%	46.08%	66.35%

Table 5.8: Grammar performance (F-score) of ERG with different lexicons

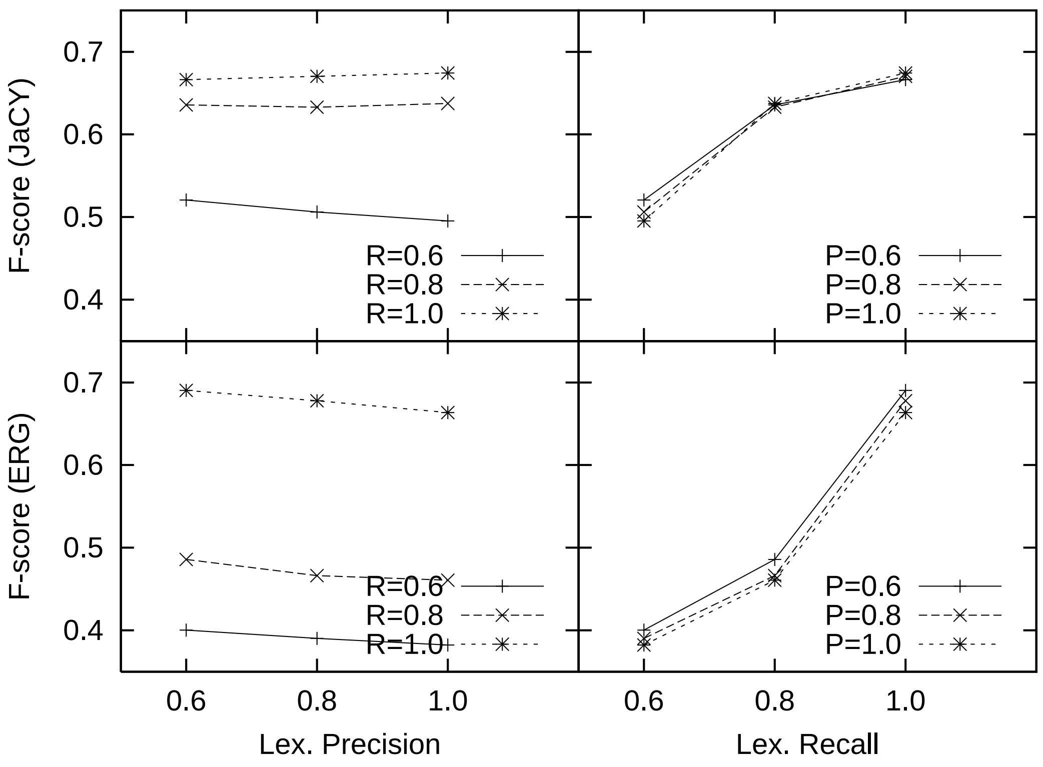


Figure 5.1: Grammar performance (F-score) with different lexicons

5.4 Discussion

5.4.1 Is F-Measure a Good Metric for DLA Evaluation?

As mentioned in earlier chapters, a number of relevant earlier works (Baldwin, 2005a; van de Cruys, 2006) have evaluated DLA results via the unweighted F-score (relative to type precision and recall). This implicitly assumes that the precision and recall of the lexicon are equally important. However, this is clearly not the case as we can see in the results of the grammar performance. For example, the lexicon configurations $060 - 100$ and $100 - 060$ of ERG (i.e., 60% precision, 100% recall vs. 100% precision, 60% recall, respectively) have the same unweighted F-scores, but their corresponding overall grammar performance (parser F-score) differs by up to 17%. Therefore, a recall-heavy interpretation of the lexicon performance is preferable from the grammar performance point of view.

In relation to the design of our DLA model, the preference in favor of recall can be realized by allowing the lexical predictor to generate larger number of entries for each instance of a candidate word.

5.4.2 Does Precision Matter?

The most interesting finding in our experiment is that the precision of the deep lexicon does not appear to have a significant impact on grammar accuracy. This is contrary to the earlier predominant belief that deep lexicons should be as accurate as possible. This belief is derived mainly from observation of grammars with relatively small lexicons. In such small lexicons, the closed-class lexical entries and frequent entries (which comprise the "core" of the lexicon) make up a large proportion of lexical entries. Hence, any loss in precision means a significant degradation of the "core" lexicon, which leads to performance loss of the grammar. For example, we find that the inclusion of one or two erroneous entries for frequent closed-class lexical type words (such as *the*, or *of* in English, for instance) may easily "break" the parser.

However, in state-of-the-art broad-coverage deep grammars such as JaCY and ERG, the lexicons are much larger. They usually have more or less similar "cores" to the smaller lexicons, but with many more open-class lexical entries and less frequent entries, which compose the "peripheral" parts of the lexicons. In our experiment, we found that more than 95% of the lexical entries belong to the top 5% of the open-class lexical types. The bigger the lexicon is, the larger the proportion of lexical entries that belongs to the "peripheral" lexicon.

In our experiment, we have only changed the "peripheral" lexicon by creating/removing lexical entries for less frequent lexemes and open-class lexical types, leaving the "core" lexicon intact. Therefore, a more accurate interpretation of the experimental results is that the precision of the *open type* and *less frequent* lexical entries does not have a large impact on the grammar performance, but their recall has a crucial effect on grammar coverage.

The consequence of this finding is that the balance between precision and recall in the deep lexicon should be decided by their impact on the task to which the grammar is applied. In research on automated DLA, the motivation is to enhance the robustness/coverage of the grammars. This work shows that grammar performance is very robust over the inevitable errors introduced by the DLA, and that more emphasis should be placed on recall.

Again, caution should be exercised here. We do *not* mean that by blindly adding lexical entries without worrying about their correctness, the performance of the grammar will be monotonically enhanced – there will almost certainly be a point at which noise in the lexicon swamps the parse chart and/or leads to unacceptable levels of spurious ambiguity. Also, the balance between precision and recall of the lexicon will depend on various expectations of the grammarians/lexicographers, i.e., the linguistic precision and generality, which is beyond the scope of this chapter.

As a final word of warning, the absolute grammar performance change that a given level of lexicon type precision and recall brings about will obviously depend on the grammar. In looking across two grammars from two very different languages, we are confident of the robustness of our results (at least for grammars of the same ilk) and the conclusions that we have drawn from them. For any novel grammar and/or formalism, however, the performance change should ideally be quantified through a set of experiments with different lexicon configurations, based on the procedure outlined here. Based on this, it should be possible to find the optimal balance between the different lexicon evaluation metrics.

5.4.3 Other Measurements for Grammar Performance

In this experiment, we have measured the grammar performance on two aspects: the parser coverage and parser accuracy. Although these are the most important performance measurements when the parsing task is concerned, there are other measurements which we have not investigated in this experiment. Parser efficiency is one of such measurements.

Unlike the context-free parsing which has a polynomial time solution, the constraint-based parsing problem is essentially NP-complete. Without extra techniques, the lexicon precision will have a direct impact on the parser performance. The less precise the lexicon is, the more entries that will be activated during the parsing. Although we have shown that the grammar rules have fairly good ability of preventing erroneous entries entering the final parsing results, higher lexical ambiguity involves more computation at local level. And occasionally, the erroneous entries trigger recursive unary rules, and lead to performance breakdown.

There has been other research (cf., van Noord, 2006) pointing out that the impact of high lexical ambiguity on parser efficiency can be largely diminished with an HMM `PoS` tagger trained on the parser output. The tagger helps select relevant lexical entries so that the improbable erroneous entries are not used for parsing. It has been shown that such techniques are able to greatly reduce lexical ambiguity without an observable decrease in parsing accuracy. A more recent study (Matsuzaki et al., 2007) also showed that supertagging techniques can be used to significantly improve efficiency without compromising accuracy.

Both of these can be connected to our study, and lead to a confident conclusion that the efficiency drawback of having higher lexical ambiguity can be largely controlled by extra preprocessing mechanism.

5.4.4 Generalization to Other Formalisms

Our experiment was carried out with DELPH-IN HPSG grammars. However, the conclusion we have reached can presumably be applied to other similar implementation platforms or even formalisms. Of course, extra experiments need to be carried out to verify this claim, which is far beyond the scope of this dissertation.

It should also be noted that in arriving at this conclusion, we assume that there is a clear distinction between the grammar and the lexicon. This applies to formalisms like HPSG, LFG, etc. This is different to strictly lexicalized formalisms like LTAG and CCG, where essentially all linguistic description resides in individual lexical entries in the lexicon. The manually compiled grammars in our experiment are also intrinsically different to grammars automatically induced from treebanks (e.g., those used in the Charniak parser (Charniak, 2000) or the various CCG parsers (Hockenmaier, 2006)). These differences sharply distinguish our work from previous research on the interaction between lexical acquisition and parse performance.

5.5 Summary

In this chapter, we have investigated the relationship between evaluation metrics for deep lexical acquisition and grammar performance. With simulated lexical acquisition results for two large scale HPSGs, we measured their effect on grammar performance in parser coverage and parser accuracy. The results show that traditional DLA evaluation based on F-measure is not reflective of grammar performance.

The precision of the lexicon appears to have minimal impact on grammar accuracy, and therefore recall should be emphasized more greatly in the design of deep lexical acquisition techniques.

6 Efficient Partial Parsing

Better be ignorant of a matter than half know it.

— Publilius Syrus *(~100 BC), Maxims*

We have spent chapters discussing techniques which can help improve the robustness of deep processing on the lexicon side. This is motivated by findings indicating that a large proportion of lack of coverage errors are related to the incompleteness of the lexicon. Most of the modern deep grammars are very much lexicalized, meaning that the grammar contains relatively few highly generalized linguistic principles and a massive amount of detailed variations are instantiated in the lexicon. While the general linguistic rules are usually well attested, the lexicon receives less harsh validation, making them vulnerable to errors when faced with free texts.

The assumption that general linguistic principles are more reliable is based on the hypothesis that all inputs are well-formed grammatical sentences, as the deep grammars we study here rely on strict binary grammaticality judgments. When the input contains mildly ungrammatical sentences, it can sometimes escape the generality of the linguistic rules. In this chapter, we will discuss a more general approach to achieve better robustness with deep grammars. Section 6.1 discusses the desirable partiality in deep processing. Section 6.2 proposes the partial parse selection models which help maximally recover intermediate results from a failed parse. Section 6.4 further discusses the efficiency concerns about different partial parse selection models, followed by Section 6.5 with in-depth discussion of the design and evaluation of the selective unpacking algorithm in the context of efficient partial parsing.

6.1 Partiality in Deep Processing

6.1.1 Why Do We Need Partiality?

Being a highly complicated symbolic rule system, a deep grammar makes unambiguous grammaticality judgments on its inputs. While each grammatical sentence receives one or more detailed analyses with rich linguistic information, there is an extreme inequality on the ungrammatical sentences: no results are available for sentences which receive no full analysis.

The strict grammaticality judgment is a desirable feature, for it allows for more precise modeling of the language. In fact, this is one of the fundamental differences between precision grammars and treebank grammars. With this feature, the grammar is not only suitable for parsing, but for other applications, like text generation (from semantics) or grammar checking, as well.

However, an accurate modeling of the language also brings other consequences to the robustness problem. For traditional precision grammars which make binary grammaticality judgments, there are two aspects of problems.

On the one hand, the binary grammaticality judgment does not accurately reflect the acceptability of inputs. Various psycholinguistic studies have shown that humans usually take a gradient grammaticality judgment. Also, human speakers/writers do produce ungrammatical sentences due to various competence and performance reasons. Given the context and common knowledge between the speakers (or writers/readers), this is not really a big problem for human/human communication. However, in deep processing the lack of flexibility in grammaticality judgment leads to serious difficulties due to the lack of extra-linguistic knowledge.

On the other hand, no analyses are given for ungrammatical sentences, at all. This is obviously against the cognitive nature of human/human interaction. Although the grammaticality level does influence the efficiency of communication, humans behave quite robustly to grammar errors. When grounded in a conversation scenario, most of the mildly ungrammatical sentences can be comprehended without problem.

Both of the aspects indicate that with conventional deep processing based on precision grammars it is difficult to achieve high levels of robustness. While adopting the gradient grammaticality in deep grammars probably entails major change in the fundamental formalism, we hope to maximally recover the most plausible and useful partial analyses from failed parses with the deep grammar largely intact.

There are three major questions to be answered in this thread of thinking. How is the partiality represented? What type of partiality is desirable? How can good partiality be retrieved efficiently? We will go through the first question in the context of bottom-up chart parsing, and leave the rest to the following sections.

6.1.2 Partiality in Bottom-Up Chart Parsing

Among various parsing algorithms, the chart parser is one of the most well studied and widely used parsers. The algorithm involves a data structure named "chart" to record the intermediate searching goals. With the chart, the algorithm eliminates backtracking and prevents a combinatorial explosion. If used together with a priority queue (usually called *agenda*), the searching progress can be guided with a priority score, and achieves n-best parsing. Various variants of the chart parsing algorithm exist. For example, Earley (1970) avoids duplication of parse items by maintaining pointers to alternative derivations in association with the item. The algorithm is usually used for parsing with context-free grammars, but can be extended for other grammar formalisms, as well.

For constraint-based formalisms (e.g., HPSG, LFG), the bottom-up chart parser works essentially unchanged, only that the items on the chart correspond to more informative data structures. For HPSG, each item on the parsing chart corresponds to a typed feature structure. For efficiency considerations, the TFSes are not stored as are on the chart, but they can be reconstructed on request.

When the grammar has one or more full analyses of the input, there will be at least one passive edge on the chart which spans the entire input. And the corresponding `TFS` satisfies specific *root* constraints set by the grammar.

When the grammar has no full analysis for the input, the parser terminates when there are no more new items which can be derived. At this stage, all the intermediate analyses are recorded on the chart as passive edges. If properly selected, these intermediate results provide rich linguistic information. Especially, due to the use of combinational semantics, the semantic fragments for the corresponding chart items can also be recovered. In this dissertation, we use the passive chart items/edges to represent the partial analyses. More specifically, we use the term *Partial Parse* to describe a set of passive parsing edges whose spans (beginning and ending positions) are non-overlapping between each other, and together they cover the entire input sequence (i.e., no skipped input tokens).

In a graph view, the intermediate results of a chart parser can be described as a directed graph, where all positions between input tokens/words are vertices, and all the passive edges derived during parsing are the directed graph arcs. Obviously such a graph is acyclic and therefore topologically sorted. A partial parse is then a path from the source vertex (the beginning position of the input) to the terminal vertex (the end position of the input).

Suppose the parsing chart consists of the edges shown in Figure 6.1, there are in total 4 possible partial parses: $\{a, b, c, d\}$, $\{a, b, f\}$, $\{a, e, d\}$ and $\{a, g\}$.

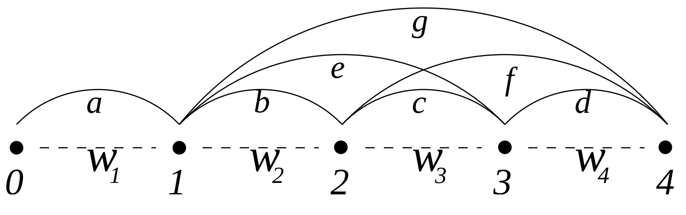


Figure 6.1: Graph representation of intermediate chart parsing results

Note that each passive edge is a sub-structure licensed by the grammar. A derivation tree or TFS can be reconstructed for it, if required. This definition of *partial parse* is effectively similar to the view of partial analyses in Kasper et al. (1999).

It should also be noted that the partial parses can be defined in other ways if using different parsing algorithms. For instance, in head-corner or left-corner parsers, the spine trees can be taken as basic units to represent partialities, as well. But in this dissertation, we restrict ourselves to the representation of very simple local analysis units.

6.2 Partial Parse Selection

Having defined the *partial parse* in the previous section, we move on to the more challenging question: which partiality is desirable?

Due to the ambiguous nature of human language, a large amount of partial parses exist for a given input. For deep linguistic processing, a high level of local ambiguity means there are even more partial parses due to the combinatorial explosion. For example, in HPSG parsing with the ERG grammar, there are usually tens to hundreds thousands of passive edges for inputs of moderate length (i.e., around 15 words). Different measures can be used to decide which one of them is preferable. For instance, various heuristics metrics are normally used, including the size (number of passive edges) of the partial parse, the size of the longest spanning edge, or a heuristically weighted path length of the partial parse, etc. More sophisticated approaches involve scoring the partial parses with a trainable statistical model. We will use the term *partial parse selection* for the task of selecting the most plausible partial parse from all the possible ones with the pool of passive edges on the parsing chart.

In this section, we will go through several partial selection models, with either pure heuristic rules, or statistical disambiguation models.

It should be noted that only linguistic plausibility is concerned in this context of discussion. We do not deny that in real applications, the domain specific knowledge also influences the preference of the selection, similar to its influence on the full parse selection. These can be hopefully also handled by adapting the parse selection models to domain specific data.

6.2.1 Longest Edge

One of the simplest and most commonly used criteria in selecting the best partial parse is to prefer the partial parses which contain an edge that covers the largest fragment of the input. For example, under such a criterion, the best partial parse in Figure 6.1 will be $\{a, g\}$, since edge g has the largest span. The logic behind this criterion is that such largest fragments should preserve the most interesting linguistic analysis of the input. As added incentive, finding the longest edge does not involve much search.

The limitations of such an approach are obvious. There is no guarantee that the longest edge will be significantly better than shorter edges, or that it will even correspond to a valid constituent. Moreover, when there are multiple edges with the same length (which is often the case in parsing), the criterion does not suffice for the choice of the best partial parse.

6.2.2 Minimum Number of Fragments

Another intuitive criterion is to prefer the partial analysis which has a minimum number of non-overlapping passive edges. Using this criterion with Figure 6.1, the best partial parse will still be $\{a, g\}$, as it comprises a minimum of two passive edges. The intuition behind is that when the full analysis is not available, the input should be broken into as a small number of fragments as possible.

The problem with this criterion is similar to the longest edge approach: on the one hand, there is no guarantee that the minimum number of edges makes the partial parse more plausible; on the other hand, it does not fully discriminate the partial parses with the same number of passive edges.

6.2.3 Shortest Path

Kasper et al. (1999) proposed an alternative solution to the problem. If the preference of each edge as a part of the partial parse can be quantitatively decided as a weight of the edge (with smaller weights assigned to better candidates), then the problem of finding the best partial parse is to find the shortest path from the start vertex to the end vertex. Since the graph is completely connected (by the lexical edges spanning all the input tokens) and topologically sorted, such a path always exists. The discovery of such a path can be done in linear time ($O(|V| + |E|)$) with the DAG-shortest-path algorithm (Cormen et al., 1990). Though not explicitly pointed out by Kasper et al. (1999), such an algorithm allows the weights of the edges to be of any real value (no assumption of positive weights) as long as the graph is a Directed Acyclic Graph (DAG).

Kasper et al. (1999) did point out that the weights of the edges can be assigned by an estimation function. For example, the implementation of the algorithm in PET preferred phrasal edges over lexical edges. Other types of edges are not allowed in the partial parse. Suppose that we assign weight 1 to phrasal edges, 2 to lexical edges, and inf to all other edges. Then for the graph in Figure 6.2, the best partial parses are $\{e, g\}$ and $\{f, g\}$, both of which have the path length of 2. It should be noted that such an approach does not always favor the paths with the longest edges (i.e., path $\{h, d\}$ is not preferred in the given example).

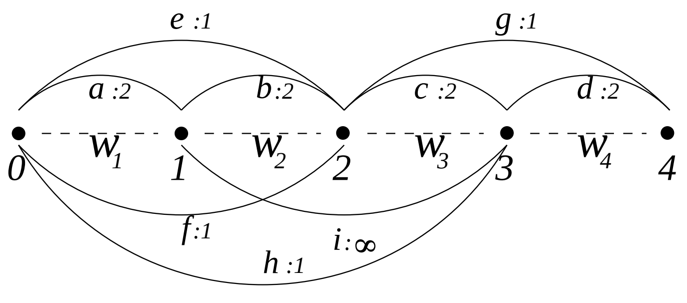


Figure 6.2: Shortest path partial parses with heuristically assigned edge weights

However, Kasper et al. (1999) did not provide any sophisticated estimation functions based on the shortest path approach. Using the heuristic weight described above, usually thousands of different paths are found with the same weight. Kasper et al. (1999) rely on another scoring function in order to re-rank the partial parses. Although different requirements for the scoring function are discussed, no further details have been defined.

Nevertheless, the shortest path approach and its variants are widely in use in many robust deep parsing systems. For instance, Riezler et al. (2002) use the *fewest chunk* method to choose the best fragment analyses for sentences without full analysis. The well-formed maximal projections (e.g., VP, NP, PP) are preferred over other grammatical categories. With this partial parse selection method, the grammar achieves full coverage on unseen data. A similar approach is also used by van Noord et al. (1999).

The approach performs reasonably well in practice. And some evaluations have been presented showing the overall parser performance given the shortest path selection (or a similar heuristic weight based approach) in the aforementioned studies. However, we find that the story about the partial parse selection method is somewhat short. In particular, it is not clear whether there are better models which, though less studied, will significantly outperform the current simple approaches. Moreover, there is a lack of systematic comparative studies of the partial parse selection model in a convincing way to show whether there is a significant performance difference among different models.

6.2.4 Alternative Estimation Functions

Generally speaking, the weights of the edges in the shortest path approach represent the quality of the local analyses and their likelihood of appearing in the analysis of the entire input.

This is an interesting parallel to the parse selection models for the full analyses, where a goodness score is usually assigned to the full analysis. As mentioned in earlier chapters already, the parse disambiguation model described by Toutanova et al. (2002) uses a maximum entropy approach to model the conditional probability of a parse for a given input sequence $P(t|w)$. Similar approaches have also been reported by Abney (1997); Johnson et al. (1999); Riezler et al. (2002); Malouf and van Noord (2004).

The main difference is that we want to rank the intermediate parsing results rather than full analyses here. There are usually some well-formedness constraints given by the grammar (e.g., *root conditions*) which must be satisfied by the maximal projections to be licensed as full analyses. But for intermediate results, there are no such constraints. On the one hand, this allows maximal robustness, for all the local analyses are available on the parsing chart without constraints from larger contexts. On the other hand, this also raises the difficulty of fully discriminating the ambiguities, for a much larger number of possible intermediate results need to be ranked.

Formally, for a given partial parse $\Phi =< t_1, \ldots, t_k >$, $\delta =< \omega_1, \ldots, \omega_k >$ is a segmentation of the input sequence so that each local analysis $t_i \in \Phi$ corresponds to a sub-string $\omega_i \in \delta$ of the input sequence ω. Therefore, the probability of the partial parse Φ given an input sequence ω is:

$$P(\Phi|\omega) = P(\delta|\omega) \cdot P(\Phi|\delta) \tag{6.1}$$

With the assumption that $P(t_i|\omega_i)$ are mutually independent for different i, we can derive:

$$P(\Phi|\omega) \approx P(\delta|\omega) \cdot \prod_{i=1}^{k} P(t_i|\omega_i) \tag{6.2}$$

Therefore, the log-probability will be

$$\log P(\Phi|\omega) \approx \log P(\delta|\omega) + \sum_{i=1}^{k} \log P(t_i|\omega_i) \tag{6.3}$$

Equation 6.3 indicates that the log-probability of a partial parse for a given input is the sum of the log-probability of local analyses for the sub-strings, with an additional component $-\log P(\delta|\omega)$ representing the conditional log-probability of the segmentation. If we use $-\log P(t_i|\omega_i)$ as the weight for each local analysis, then the DAG shortest path algorithm will quickly find the partial parse that maximizes $\log P(\Phi|\omega) - \log P(\delta|\omega)$.

The probability $P(t_i|\omega_i)$ can be modeled in a similar way to the maximum entropy based full parse selection models:

$$P(t_i|\omega_i) = \frac{\exp \sum_{j=1}^{n} \lambda_j f_j(t_i, \omega_i)}{\sum_{t' \in T} \exp \sum_{j=1}^{n} \lambda_j f_j(t', \omega_i)} \tag{6.4}$$

where T is the set of all possible structures that can be assigned to ω_i, $f_1 \dots f_n$ are the features and $\lambda_1 \dots \lambda_n$ are the parameters. The parameters can be efficiently estimated from a treebank, as shown by Malouf (2002). The only difference from the full parse selection model is that here intermediate results are used to generate events for training the model (i.e., the intermediate nodes are used as positive events, if they occur on one of the active trees, or as negative events, if not). Since there is a huge number of intermediate results available, we only randomly select a part of them as training data. This is essentially similar to the approach of Osborne (2000), where there is an infeasibly large number of training events, only part of which is used in the estimation step. The exact features used in the log-linear model can significantly influence the disambiguation accuracy. However, this is beyond the scope of this discussion. In this experiment we used the same features as those used in the `PCFG-S` model of Toutanova et al. (2002) (i.e., depth-1 derivation trees).

With the n-gram language models trained on a large corpus, we can derive the probability of input sequence $P(\omega)$, as well as all the sub-sequences in the segmentations $P(\omega_i)$. To estimate the conditional probabilities of the segmentations, we first use the following estimation to derive the unconditioned probabilities.

$$\hat{P}(\delta) = \prod_{i=1}^{k} P(\omega_i) \tag{6.5}$$

The estimation of the conditional probability can be derived by normalizing the unconditioned probabilities over all the possible segmentations for the input:

$$\hat{P}(\delta|\omega) = \frac{\hat{P}(\delta)}{\sum_{\delta' \in \Delta} \hat{P}(\delta')} \tag{6.6}$$

where Δ indicates the set of all possible segmentations. A closer look easily reveals that $\hat{P}(\delta|\omega)$ is solely determined by how independent the occurrences of word groups are around each segmentation point. Considering a bi-gram language model, the segmentation probability is changed by a factor of $\frac{P(w_i)\cdot P(w_{i+1})}{P(w_i, w_{i+1})}$ from the language model probability of the input sequence $P(\omega)$ for each segmentation point at i. Intuitively, a good (plausible and probable) segmentation should separate the input sequence at points where the joint probabilities are lower than the product of individual probabilities. This indicates that the words around the segmentation points are less correlated. Also, note that since $\hat{P}(\delta|\omega)$ will be normalized, the computation of the language model probability for the input sequence is not necessary. Computational-wise, the worst case time complexity of computing $\hat{P}(\delta|\omega)$ for all segmentations is $O(|\Delta| \cdot |\omega|)$. $|\Delta|$ can be potentially large, for each position between words can be considered as a segmentation point, leading to a total number of different segmentations up to $2^{|\omega|-1}$. Fortunately, in practice not all of them are licensed by the grammar.

Unfortunately, the shortest path algorithm itself is not able to directly find the maximized $P(\Phi|\omega)$, for each passive edge can occur in different segmentations, making the assignment of $P(\delta|\omega)$ to edges difficult. Fully searching all the paths can be computationally expensive when there are a lot of different segmentations. In order to achieve a balance between accuracy and efficiency, two different approximation approaches are taken.

One way is to assume that the component $\log P(\delta|\omega)$ in Equation 6.3 has less significant effect on the quality of the partial parse. If this is valid, then we can simply use $-\log P(t_i|\omega_i)$ as edge weights, and use the shortest path algorithm to obtain the best Φ. This will be referred to as *model I*.

An alternative way is to first retrieve several "good" δ with relatively high $P(\delta|\omega)$, and then select the best edges t_i that maximize

$P(t_i|\omega_i)$ for each ω_i in δ. We call this approach the *model II*.

How well these strategies work will be evaluated in Section 6.3. Other strategies or more sophisticated searching algorithms (e.g., the genetic algorithm) can also be used, but we will leave that to future research. It is even possible to do a complete search for a global optimal partial parse, though with even higher (potentially exponential) computational complexity.

6.3 Evaluation of Partial Parse Selection Models

The evaluation of partial parses is not as easy as the evaluation of full parses. For full parsers, there are generally two ways of evaluation. For parsers that are trained on a treebank using an automatically extracted grammar, an unseen set of manually annotated data is used as the test set. The parser output on the test set is compared to the gold standard annotation, either with the widely used *PARSEVAL*[1] measurement, or with more annotation-neutral dependency relations. The evaluation procedure is largely automated. The annotation quality plays a dominating role in such evaluation. When the analysis of a specific language phenomenon needs to be changed, the treebank annotation needs to be updated accordingly. It is, to say the least, difficult and time consuming for manually annotated treebanks.

For parsers based on manually compiled precision grammars, more human judgment is involved in the evaluation. Unlike the treebank induced grammars, the precision grammar based parsing output does not conform to the existing treebank annotation. Moreover, the analysis can change dramatically with the evolution of the grammar. Therefore, the *PARSEVAL* metrics are not practical for manually compiled precision grammars. More annotation-neutral

[1]The PARSEVAL metric counts the proportion of bracketings which group the same sequences of words in both the gold standard trees and the parser output. Early versions of the PARSEVAL metric ignored the question whether matching sequences were labelled the same way in both trees (aka. unlabelled PARSEVAL), but more refined versions have subsequently taken this into account (aka. labelled PARSEVAL).

evaluation methods (e.g., dependency relation-based evaluation) are plausible. However, extra effort is needed to convert the outputs, if the precision grammar uses a different representation. For the DELPH-IN grammars, we use the (Robust) Miminum Recursion Semantics ((R)MRS; Copestake et al. (1999); Copestake (2006)) for semantic output. The conversion from MRS to dependency structure is, though possible, less well-studied.

Instead of relying on pre-annotated gold standard treebanks, the evaluation of manually compiled DELPH-IN precision grammars is accomplished by so called dynamic treebanks. With the evolution of the grammar, the treebank, as the parsing output from the grammar, changes over time (Oepen et al., 2002). The grammar writer needs to update the treebank by inspecting the parses generated by the grammar and either "accepts" or "rejects" the new analyses. And the performance change of the grammar/parser is evaluated upon the updated treebank: the coverage is the proportion of grammatical sentences which receive at least one correct analysis; the overgeneration is the proportion of ungrammatical sentences which receive at least one analysis, etc. However, the size of the dynamic treebanks is usually relatively small, for the burden of updating the entire treebank after each grammar change is non-trivial. Also, the fixed set of test items make the test non-blind. After several iterations, the grammar can be specifically tuned for the test set, either intentionally or unintentionally. Therefore, only the first round of treebanking can be regarded as an unbiased evaluation. In the evaluation of parser accuracy, the performance of the statistical disambiguation model should also be considered together with the grammar performance.

However, the evaluation becomes more difficult for partial parsing. In order to evaluate the partial parsing results for manually compiled grammars, the criterion for acceptable analyses becomes less evident. And most current treebanking tools are not designed for annotating partial analyses. Large-scale manually annotated treebanks do have the annotation for sentences that deep grammars are not able to fully analyze. But the annotation difference in other language resources makes the comparison less straightforward. More complication is involved with the platform and resources used in this experiment. For instance, the transformation of incomplete RMRS fragments into other representations is largely an open question.

In this section, we use both manual and automatic evaluation methods on the partial parsing results. Different processing resources are used to help the evaluation from the syntactic, as well as the semantic point of view. Some of the results have been reported earlier by Zhang et al. (2007a), as well.

6.3.1 Syntactic Evaluation

In order to evaluate the quality of the syntactic structures of the partial parses, we implemented the partial parse models described in the previous section in the PET parser. The *nov-06* version of the ERG is used for the experiment. As test set, we used a subset of sentences from the Wall Street Journal Section 22 from the Penn Treebank. The subset contains 143 sentences which i) do not receive any full analysis licensed by the grammar, and ii) do not contain lexical gaps (input tokens for which the grammar cannot create any lexical edge). The first criterion allows us to investigate the potential of our partial parsing mechanism, while the second criterion avoids the complication with the coverage loss due to an incomplete lexicon. Although, the techniques developed in previous sections can largely improve the lexical coverage and provide us with a larger test set, we would like to carefully separate the different aspects of robustness in this study. The average sentence length in this test set is 24 words.

Due to the inconsistency of the tokenization, bracketing and branching between the Penn Treebank annotation and the handling in ERG[2], we manually checked the partial parse derivation trees. Each output is marked as one of the three cases: **GBL** (good labelled bracketing) if both the bracketing and the labeling of the partial parse derivation

[2]In the Penn Treebank, most of the punctuations are treated as separate tokens/words, while in the ERG most of them are treated as affixes. ERG analyses are strictly either unary or binary, while the Penn Treebank branchings are much more flexible (i.e., flat construction for less agreed upon analyses).

trees are good (with no more than two brackets crossing or four false labellings); **GB** (good unlabelled bracketing) if the bracketings of the derivation trees are good (with no more than two brackets crossing), but the labeling is bad (with more than four false labellings); or **E** (erroneous), if otherwise.

The manual evaluation results are listed in Table 6.1. The test set is processed with two models presented in Section 6.2 (**M-I** for *model I*, **M-II** for *model II*). For comparison, we also evaluate for the approach using the shortest path with heuristic weights (denoted by **SP**). In case there are more than one path found with the same weight, only the first one is recorded and evaluated.

	GBL		GB		E	
	#	%	#	%	#	%
SP	55	38.5%	64	44.8%	24	16.8%
M-I	61	42.7%	46	32.2%	36	25.2%
M-II	74	51.7%	50	35.0%	19	13.3%

Table 6.1: Syntactic evaluation results for different partial parse selection models

The results show that the naïve shortest path approach based on the heuristic weights works pretty well at predicting the bracketing (with 83.3% of the partial parses having less than two brackets crossing). But, when the labeling is also evaluated, it is worse than *model I*, and even more significantly outperformed by *model II*.

6.3.2 Semantic Evaluation

Evaluation of the syntactic structure only reflects the partial parse quality from some aspects. In order to get a more thorough comparison between different selection models, we look at the semantic output generated from the partial parses.

The same set of 143 sentences from the Wall Street Journal Section 22 of the Penn Treebank is used. The `RMRS` semantic representations are generated from the partial parses with different selection models. To compare, we used `RASP` 2 (Briscoe et al., 2006), a domain-independent robust parsing system for English. According to Briscoe and Carroll (2006), the parser achieves a fairly good accuracy of around 80%. The reasons why we choose `RASP` for the evaluation are: i) `RASP` has reasonable coverage and accuracy; ii) its output can be converted into `RMRS` representation with the `LKB` system. Since there is no large scale `(R)MRS` treebank with sentences not covered by the DELPH-IN precision grammars, we hope to use the `RASP`'s `RMRS` output as a standalone annotation to help the evaluation of the different partial parse selection models. However, we do not claim that the `RASP` output is the "gold standard" from any aspect. In fact, a shortest path algorithm similar to ours is used in the system to achieve maximal robustness. The output from `RASP` is used as reference to help us compare whether there is a significant performance difference between our partial parse selection models. But none of the following results should be taken as an absolute quantitative measure.

In future research, we do see an emerging need for a platform independent standard evaluation for deep linguistic processing systems. For both deep and shallow parsing systems, we have seen that in recent years more and more researchers have expressed a similar opinion in different ways (e.g., Carroll, 1998; Carroll et al., 2002). More recently, we have also seen that for the shallow parsing community, dependency structure based evaluation is becoming a de facto standard (Buchholz and Marsi, 2006). Some of the deep processing systems can produce compatible dependency structures, which allow cross platform evaluation. However, due to its limitation in expression power, fine grain linguistic description of subtle meanings is not always available. For the deep processing systems which adopt richer semantic representations (e.g., `(R)MRS`), conversion to dependency structures is a workaround, rather than an optimal solution. One possible direction is to explore the methods to convert from dependency structures into `(R)MRS`, and use `(R)MRS` as the basis for evaluation. It is also necessary to create a larger gold standard `(R)MRS` treebank which is manually corrected and independent from other specific language resources. Also, the development of `SEM-I` (stands for *semantic interface*) for `DELPH-IN` deep grammars is an initial step in such an direction, where the clearly defined semantic output will largely facilitate the platform independent parser evaluation.

Back to our evaluation, in order to compare the RMRS from the RASP and the partial parse selection models, we used the similarity measurement proposed by Dridan and Bond (2006). The comparison outputs a distance value between two different RMRSs. We normalized the distance value to be between 0 and 1. For each selection model, the average RMRS distance from the RASP output is listed in Table 6.2.

	RMRS Dist.(ϕ)
SP	0.674
M-I	0.330
M-II	0.296

Table 6.2: RMRS distance to RASP outputs

Again, we see that the outputs of *model II* achieve the highest similarity when compared to the RASP output. With some manual validation, we do confirm that the different similarity does imply a significant difference in the quality of the output RMRS. The shortest path with heuristic weights yielded very poor semantic similarity. The main reason is that not every edge with the same span generates the same semantics. Therefore, although the *SP* receives reasonable bracketing accuracy, it has less idea of the goodness of different edges with the same span. By incorporating $P(t_i|\omega_i)$ in the scoring model, models *I* and *II* can produce RMRSs with much higher quality.

6.4 Efficiency Concerns

In earlier chapters, we mentioned that there are three main aspects of deep processing which attract most of the research interests: robustness/coverage, specificity and efficiency. While this dissertation is mainly about robustness and coverage, the specificity problem is also discussed. For instance, the partial parse selection models discussed earlier in the chapter deal with the ambiguity of intermediate parsing results, and help select the most probable partial analyses.

The efficiency problem, as the other element from the trinity, will be discussed in this and the next section.

Over the last decades, the efficiency of deep parsing techniques has been significantly improved and matured to the degree that deep processing of large corpora with moderate personal computers is becoming practical. Unlike shallow processing, deep processing usually involves formalisms with stronger representation power, and larger searching space. To cope with the complex data structures (e.g., `TFS`es) more efficiently, various techniques have been developed. For instance, in unification-based parsing, the most time consuming operations are feature structure unifications and copies. The quasi-destructive graph unifications algorithm described by Tomabechi (1991) is based on the simple, yet important insight that not all the unifications succeed. By eliminating copying for unsuccessful unifications, the algorithm is often seen as the most efficient unification algorithm for natural language processing today.

Another useful efficient processing technique for unification-based processing is called "quick-check" (Malouf et al., 2000). It reduces the time spent on unifications by starting on the feature paths which fail most often. This technique requires training on a corpus to obtain the statistical failure frequency for different feature paths, and it is most effective when the unification failure rate is high.

Large `TFS`es also present a storage difficulty. In fact, not all the `TFS`es should necessarily be stored in memory throughout the parsing process. Therefore, a commonly used strategy is to only store the basic information of the parsing edge (i.e., the start and end position, type, parents and daughters). The feature structures are not stored on the chart, and only reconstructed on request.

Another aspect of efficient processing is concerned with the handling of local ambiguities. The searching involved in natural language processing is rooted to the fact that languages are ambiguous. Especially when looked at under the microscope, each sub-string can receive a large number of different analyses without looking at the larger context in which it is used. In bottom up chart parsing, such ambiguous local readings will be generated and kept on the chart. For deep processing, such local ambiguities are necessary in order to help discriminate the subtle different readings. However, this also leads to the inefficiency of storing redundant information and repeatedly searching over them. To deal with this problem, the ambiguity packing mechanisms are used. This makes parsing a two-phase operation. During the first phase, the local ambiguities are grouped together in a packed representation, and the parse forest is created efficiently. The second phase decomposes the packed representations and retrieves the parsing results. In constraint-based parsing, a technique called retro-active subsumption base packing has been introduced to achieve high packing ratio (Oepen and Carroll, 2000).

In relation to our partial parsing approach, the ambiguity packing mechanism is especially intriguing. In normal parsing, top down predictions (e.g., the entire input sequence to be analyzed as specific category) can be used to narrow the search space. But in our partial parsing, no constraints are presumed, and a pure bottom-up approach must be used. For large grammars, this means a huge amount of passive edges on the parsing chart. Without ambiguity packing, the parser will run out of storage space quickly with the increase of input length.

In fact, all of the efficient processing techniques mentioned above (as well as many others) have been implemented in the `PET` parser. But it was not clear until recently how to efficiently recover the best readings from the packed representation. This technique is known as *"selective unpacking"*, and will be discussed in detail in the next section.

6.5 Selective Unpacking

6.5.1 Ambiguity Packing

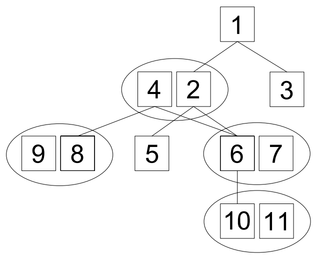
Oepen and Carroll (2000) introduced an efficient ambiguity packing algorithm for unification-based processing systems. The motivation is that the equivalence based packing mechanism is ineffective for systems using formalisms like (typed) feature structures. In practice, the chance of two (typed) feature structures being identical is very small. They might share most of the information, but still exhibit minor differences. Therefore, equivalence based ambiguity packing, though very useful in `CFG` parsing, requires refinement when faced with `TFS`es. The subsumption relation between `TFS`es provides a good substitute for the equivalency. `TFS` A subsumes B when B contains more constraints (and is more specific) than A. Whenever a unification succeeds with B, it will also succeed with A. In subsumption-based ambiguity packing, more specific edges are packed into more general ones. This helps reduce the number of unifications during the parse forest creation phase. Oepen and Carroll (2000) also provided an efficient bi-directional subsumption check algorithm with linear-time complexity. Therefore, not only new edges are packed into existing more general edges. Old edges will be also packed into newly created ones if the subsumption holds. This allows for maximal packing ratio. The empirical results reported by Oepen and Carroll (2000) show that the packing mechanism largely reduced the memory usage and parsing time. Especially during the parse forest creation phase, the average parsing time increases with the sentence length in a nice linear fashion.

6.5.2 Selective Unpacking Procedure

In Oepen and Carroll (2000), the unpacking phase of the parser was not discussed in details. It was pointed out that since more specific feature structures are packed, further unification is necessary during the unpacking phase to validate its compatibility with other siblings of the grammar rule. When unpacking exhaustively, it can be done by recursively checking all the combinations in the cross-multiplication of all the unpacked daughters of a given edge. But empirical results show (both by Oepen and Carroll (2000) and in our practice) that the unpacking time grows exponentially with the length of the input.

Carroll and Oepen (2005) presented an algorithm to selectively unpack the best readings according to the maximum entropy score assigned by the disambiguation model. The algorithm was originally proposed in the context of efficient chart-based text generation. But it is applicable to efficient parsing, as well.

Two key notions introduced in the selective unpacking procedure are the concepts of i) *decomposing* an edge locally into candidate ways of instantiating it and of ii) nested contexts of ‘horizontal’ search for ranked *hypotheses* (i.e., uninstantiated edges) about candidate sub-trees. Figure 6.3 is an example from Carroll and Oepen (2005), showing the packed parse forest and the decompositions.



$$
\begin{aligned}
\boxed{1} &\rightarrow \left\langle \boxed{2}\,\boxed{3} \right\rangle \mid \left\langle \boxed{4}\,\boxed{3} \right\rangle \\
\boxed{2} &\rightarrow \left\langle \boxed{5}\,\boxed{6} \right\rangle \mid \left\langle \boxed{5}\,\boxed{7} \right\rangle \\
\boxed{4} &\rightarrow \left\langle \boxed{8}\,\boxed{6} \right\rangle \mid \left\langle \boxed{8}\,\boxed{7} \right\rangle \mid \left\langle \boxed{9}\,\boxed{6} \right\rangle \mid \left\langle \boxed{9}\,\boxed{7} \right\rangle \\
\boxed{6} &\rightarrow \left\langle \boxed{10} \right\rangle \mid \left\langle \boxed{11} \right\rangle
\end{aligned}
$$

Figure 6.3: Sample forest and sub-node decompositions

The ovals in the forest (on the left) indicate packing of edges under subsumption, i.e., edges $\boxed{4}$, $\boxed{7}$, $\boxed{9}$, and $\boxed{11}$ are *not* in the chart proper. During unpacking, there will be multiple ways of instantiating a chart edge, each obtained from cross-multiplying alternate daughter sequences locally. The elements of this cross-product we call *decomposition*, and they are pivotal points both for stochastic scoring and dynamic programming in selective unpacking. The table on the right shows all non-leaf decompositions for our example packed forest: given two ways of decomposing $\boxed{6}$, there will be three candidate ways of instantiating $\boxed{2}$ and six for $\boxed{4}$, respectively, for a total of nine full trees.

Carroll and Oepen (2005) assume a context size of no more than depth one. Given a decomposition, i.e., a vector of candidate daughters to a token construction, an index vector $< i_0 \ldots i_n >$ serves to keep track of ‘vertical’ search among daughter hypotheses, where each index i_j denotes the i-th hypothesis of the daughter at position j. Hypotheses are associated with ME scores and ordered within each nested context by means of a local agenda (stored in the original representative edge). Given the additive nature of ME scores on complete derivations, it can be guaranteed that larger derivations including an edge e as a sub-constituent on the fringe of their local context of optimization will use the best instantiation of e in their own best instantiation. The second-best larger instantiation, in turn, will be obtained from moving to the second-best hypothesis for one of the elements in the (right-hand side of the) decomposition. Therefore, nested local optimizations result in a top-down, exact n -best search through the generation forest, and matching the 'depth' of local decompositions to the maximum required ME feature context effectively prevents exhaustive cross-multiplication of packed nodes.

The core of the algorithm is a top-down search with dynamic programming, described as pseudo-code in Figure 6.4.

The entry procedure selectively-unpack-edge() controls the enumeration of the top n best instantiated realizations for a result *edge* from a packed forest. The main procedure hypothesize-edge() controls the nested search for the n^{th} best realization (hypothesis) of a given *edge* without instantiating it. The set of the decompositions for the edge will be initialized on the first call and put onto the agenda (see lines 11 – 17). Furthermore, the procedure retrieves the current next-best hypothesis from the agenda (line 18), generates new hypotheses by advancing the daughter indices (while skipping over configurations seen earlier) and calling itself recursively for each new index (line 19 – 27), and, finally, arranges for the resulting hypothesis to be cached for later invocations on the same *edge* and i values (line 28). An auxiliary procedure decompose-edge() performs local cross-multiplication as shown in the examples in Figure 6.3. Another utility function not shown in the pseudo-code is advance-indices(), a 'driver' routine searching for alternate instantiations of the daughter edges, e.g., advance-indices($\langle 0\ 2\ 1\rangle$) $\rightarrow \{\langle 1\ 2\ 1\rangle\ \langle 0\ 3\ 1\rangle\ \langle 0\ 2\ 2\rangle\}$. Finally, instantiate-hypothesis() is the function that actually builds result trees, replaying the unifications of constructions from the grammar (as identified by chart edges) with the feature structures of daughter constituents.

Note that there is no feature structure unifications or copies in-

```
procedure selectively-unpack-edge(edge , n) $\equiv$
  results $\leftarrow \langle\,\rangle$; i $\leftarrow$ 0;
  do
    hypothesis $\leftarrow$ hypothesize-edge(edge , i); i $\leftarrow$ i + 1;
    if (new $\leftarrow$ instantiate-hypothesis(hypothesis)) then
      n $\leftarrow$ n $-$ 1; results $\leftarrow$ results $\oplus \langle$new$\rangle$;
  while (hypothesis and n $\geq$ 1)
  return results;

procedure hypothesize-edge(edge , i) $\equiv$
  if (edge.hypotheses[i]) return edge.hypotheses[i];
  if (i = 0) then
    for each (decomposition in decompose-edge(edge)) do
      daughters $\leftarrow \langle\,\rangle$; indices $\leftarrow \langle\,\rangle$
      for each (edge in decomposition.rhs) do
        daughters $\leftarrow$ daughters $\oplus \langle$hypothesize-edge(edge, 0)$\rangle$;
        indices $\leftarrow$ indices $\oplus \langle 0 \rangle$;
      new-hypothesis(edge, decomposition, daughters, indices);
  if (hypothesis $\leftarrow$ edge.agenda.pop()) then
    for each (indices in advance-indices(hypothesis.indices)) do
      if (indices $\in$ hypothesis.decomposition.indices) then continue
      daughters $\leftarrow \langle\,\rangle$;
      for each (edge in hypothesis.decomposition.rhs) each (i in indices) do
        daughter $\leftarrow$ hypothesize-edge(edge, i);
        if (not daughter) then daughters $\leftarrow \langle\,\rangle$; break
        daughters $\leftarrow$ daughters $\oplus \langle$daughter$\rangle$;
      if (daughters) then
        new-hypothesis(edge, hypothesis.decomposition, daughters, indices)
    edge.hypotheses[i] $\leftarrow$ hypothesis;
    return hypothesis;

procedure new-hypothesis(edge , decomposition , daughters , indices) $\equiv$
  hypothesis $\leftarrow$ new hypothesis(decomposition, daughters, indices);
  edge.agenda.insert(score-hypothesis(hypothesis), hypothesis);
  decomposition.indices $\leftarrow$ decomposition.indices $\cup$ {indices};
```

Figure 6.4: Selective unpacking procedure

volved in the procedure hypothesize-edge(); those computationally expensive operations are only involved in routine instantiate-hypothesis(), and performed as late as possible on top ranked hypotheses only.

Unlike the beam search based n-best forest read-out procedure proposed by Malouf and van Noord (2004), Carroll and Oepen (2005) are able to find the exact n-best readings while avoiding exhaustive cross-multiplication of packed nodes.

6.5.3 Feature Context Extension

The main problem with Carroll and Oepen (2005)'s selective unpacking algorithm is that its search context is limited to the local trees of depth one. However, some of the ME model features used in the current system require a larger context to be observed. In Table 6.3, we list some examples of the features used in our disambiguation model. The *Type* column indicates the template corresponding to each sample feature. Type 1 is for the CFG style local branching features. Type 2 is similar to type 1, only that only one of the daughters of the branching is listed. The integer that starts each feature of type 1 or 2 indicates the degree of grandparenting. Type 3 is for the lexical n-gram features, with the first integer indicating the size. The symbols $\triangle$ and $\lhd$ denote the root of the tree and left periphery of the yield, respectively.

Using contexts of trees with depth one, only those features of type 1 and 2 with grandparenting level 0 can be computed. Although the authors claimed that the extension to larger context is straightforward, no concrete solution was given in the paper.

Both Toutanova et al. (2005) and our own experiments (described later in the section) suggest that properties of larger contexts and especially grandparenting can greatly improve parse selection accuracy. The following paragraphs outline how to generalize the basic selective unpacking procedure, while retaining its key properties: exact n-best enumeration with minimal search. This work has been partly reported earlier in Zhang et al. (2007b), as well.

Our generalization of the algorithm distinguishes between 'upward'

Type	Sample Features
1	⟨0 subjh hspec third_sg_fin_verb⟩
1	⟨1 △ subjh hspec third_sg_fin_verb⟩
1	⟨0 hspec det_the_le sing_noun⟩
1	⟨1 subjh hspec det_the_le sing_noun⟩
1	⟨2 △ subjh hspec det_the_le sing_noun⟩
2	⟨0 subjh third_sg_fin_verb⟩
2	⟨0 subjh hspce⟩
2	⟨1 subjh hspec det_the_le⟩
2	⟨1 subjh hspec sing_noun⟩
3	⟨1 n_intr_le *dog*⟩
3	⟨2 det_the_le n_intr_le *dog*⟩
3	⟨3 ◁ det_the_le n_intr_le *dog*⟩

Table 6.3: Examples of structural features

contexts, with grandparenting with dominating nodes as a representative feature type, and 'downward' extensions, which we discuss for the example of lexical n-gram features (type 3 features in Table 6.3).

A naïve approach to selective unpacking with grandparenting might be extending the cross-multiplication of local ambiguity to trees of more than depth one. However, with multiple levels of grandparenting this approach would greatly increase the combinatorics to be explored, and it would pose the puzzle of overlapping local contexts of optimization. Choices made among the alternates for one packed node would interact with other ambiguity contexts in their internal nodes, rather than merely at the leaves of their decompositions. However, it is sufficient to keep the depth of decompositions to minimal sub-trees and rather contextualize each decomposition as a whole. Assuming our sample forest and set of decompositions from Figure 6.3, let $\langle\boxed{1}\boxed{4}\rangle : \boxed{6} \rightarrow \langle\boxed{10}\rangle$ denote the decomposition of node $\boxed{6}$ in the context of $\boxed{4}$ and $\boxed{1}$ as its immediate parents. When descending through the forest, hypothesize-edge() can, without significant extra cost, maintain a vector $\vec{P} = \langle p_n \; \ldots \; p_0\rangle$ of parents of the current node, for n-level grandparenting. For each packed node, the bookkeeping elements of the graph search procedure need to be contextualized on $\vec{P}$, viz. (a) the edge-local priority queue, (b) the record of index vectors hypothesized already, and (c) the cache of previous instantiations. Assuming each is stored in an associative array, then all references to edge.agenda in the original procedure can be replaced by edge.agenda[$\vec{P}$], and likewise for other slots. With these extensions in place, the original control structure of nested, on-demand creation of hypotheses and dynamic programming of partial results can be retained, and for each packed node with multiple parents ($\boxed{6}$ in our sample forest) there will be parallel, contextualized partitions of optimization. Thus, extra combinatorics introduced in this generalized procedure are confined to only such nodes, which (intuitively at least) appear to establish the lower bound of added search needed—while keeping the algorithm non-approximative. Empirical data on the degradation of the procedure in growing levels of grandparenting and the number of n-best results to be extracted from the forest is to be shown later in the section.

Finally, we turn to enlarged feature contexts that capture information from nodes *below* the elements of a local decomposition. Consider the example of feature type 3 in Table 6.3, n-grams (of various size) over properties of the yield of the parse tree. For now we only consider lexical *b*i-grams. For an edge e dominating a sub-string of n words $\langle w_i \ldots w_{i+n-1} \rangle$ there will be $n - 1$ bi-grams internal to e, and two bi-grams that interact with w_{i-1} and w_{i+n}—which will be determined by the left- and right-adjacent edges to e in a complete tree. The internal bi-grams are unproblematic, and we can assume that ME weights corresponding to these features have been included in the sum of weights associated to e. Seeing that e may occur in multiple trees, with different sister edges, the selective unpacking procedure has to take this variation into account when evaluating local contexts of optimization.

Let ${}_{x}e_{y}$ denote an edge e, with x and y as the lexical types of its leftmost and rightmost daughters, respectively. Returning to our sample forest, assume lexicalizations ${}_{\beta}\boxed{10}_{\beta}$ and ${}_{\gamma}\boxed{11}_{\gamma}$ (each spanning only one word), with $\beta \neq \gamma$. Obviously, when decomposing $\boxed{4}$ as

$\langle \boxed{8}\boxed{6} \rangle$, its ME score, in turn, will depend on the choice made in the expansion of $\boxed{6}$: the sequences $\langle {}_{\alpha}\boxed{8}_{\alpha}\, {}_{\beta}\boxed{6}_{\beta} \rangle$ and $\langle {}_{\alpha}\boxed{8}_{\alpha}\, {}_{\gamma}\boxed{6}_{\gamma} \rangle$ will differ in (at least) the scores associated with the bi-grams $\langle \alpha\, \beta \rangle$ vs. $\langle \alpha\, \gamma \rangle$. Accordingly, when evaluating candidate decompositions of $\boxed{4}$, the number of hypotheses that need to be considered is doubled; as an immediate consequence, there can be up to eight distinct lexicalized variants for the decomposition $\boxed{1} \rightarrow \langle \boxed{4}\boxed{3} \rangle$ further up in the tree. It may look as if combinatorics will cross-multiply throughout the tree—in the worst case returning us to an exponential number of hypotheses—but this is fortunately not the case: regarding the external bi-grams of $\boxed{1}$, node $\boxed{6}$ no longer participates in its left- or rightmost periphery, so variation internal to $\boxed{6}$ is not a multiplicative factor at this level. This is essentially the observation of Langkilde (2000), and her bottom-up factoring of n-gram computation is easily incorporated into our top-down selective unpacking control structure. At the point where hypothesize-edge() invokes itself recursively (line 23 in Figure 6.4), its return value is now a set of lexicalized alternates, and the hypothesis creation (in line 26) can take into account the local cross-product of all such alternation. Including additional properties from non-local sub-trees (for example higher-order n-grams and head lexicalization) is a straightforward extension of this scheme, replacing our per-edge left- and rightmost periphery symbols with a generalized vector of externally relevant, internal properties. In addition to traditional (head) lexicalization as we have just discussed it, such extended 'downward' properties on decompositions—percolated from daughters to mothers and cross-multiplied as appropriate—could include metrics of constituent weight too, for example in order to enable the ME model to prefer "balanced" coordination structures.

However, given that Toutanova et al. (2005) obtain only marginally improved parse selection accuracy from the inclusion of n-gram (and other lexical) ME features, we have left the implementation of lexicalization and empirical evaluation for future work.

6.5.4 Instantiation Failure Caching and Propagation

As we pointed out earlier, during the unpacking phase, unification is only replayed in instantiate-hypothesis() on the top-level hypotheses. It is only at this step that inconsistencies in the local combinatorics are discovered. However, such a discovery can be used to improve the unpacking routine by (a) avoiding further unification on hypotheses that have already failed to instantiate, (b) avoiding creating new hypotheses based on failed sub-hypotheses. This requires some changes to the routines instantiate-hypothesis() and hypothesize-edge(), as well as an extra boolean marker for each hypothesis.

The extended instantiate-hypothesis() starts by checking whether the hypothesis is already marked as failed. If it is not marked so, the routine recursively instantiates all sub-hypotheses. Any failure will again lead to instant return. Otherwise, unification is used to create a new edge from the outcome of the sub-hypothesis instantiations. If this unification fails, the current hypothesis is marked. Moreover, all its ancestor hypotheses are also marked (by recursively following the pointers to the direct parent hypotheses) as they are also guaranteed to fail.

Correspondingly, hypothesize-edge() needs to check the instantiation failure marker to avoid returning hypotheses that are guaranteed to fail. If a hypothesis coming out of the agenda is already marked as failed, it will be used to create new hypotheses (with advance-indices()), but dropped afterward. Subsequent hypotheses will be popped from the agenda until either a hypothesis that is not marked as failed is returned, or the agenda is empty.

Moreover, hypothesize-edge() also needs to avoid creating new hypotheses based on failed sub-hypotheses. When a failed sub-hypothesis is found, the creation of the new hypothesis is skipped. But the index vector $\vec{I}$ may not be simply discarded. Otherwise hypotheses based on advance-indices($\vec{I}$) will not be reachable in the search. On the other hand, simply adding every advance-indices($\vec{I}$) on to the pending creation list is not efficient either in the case where multiple sub-hypotheses fail.

To solve the problem, we compute a failure vector $\vec{F} = \langle f_0 \ldots f_n \rangle$, where f_j is 1 when the sub-hypothesis at position j is known as failed, and 0 otherwise. If a sub-hypothesis at position j fails, then all the index vectors having value i_j at position j must also fail. By putting the result of $\vec{I} + \vec{F}$ on the pending creation list, we can safely skip the failed rows of sub-hypotheses, while not losing the reachability of the others. As an example, suppose we have a ternary index vector $\langle 3\,1\,2 \rangle$ for which a new hypothesis is to be created. By checking the instantiation failure marker of the sub-hypotheses, we find that the first and the third sub-hypotheses are already marked. The failure recording vector will then be $\langle 1\,0\,1 \rangle$. By putting $\langle 4\,1\,3 \rangle = \langle 3\,1\,2 \rangle + \langle 1\,0\,1 \rangle$ on to the pending hypothesis creation list, the failed sub-hypotheses are skipped.

For completeness, we give the pseudo-code of the extended algorithm with support for arbitrary levels of grandparenting features and instantiation failure caching in Figure 6.5.

The main difference to the original algorithm is that we added an extra parameter *path* to the procedure hypothesize-edge() to discriminate different grandparenting context. Multiple agendas are used to control the search of the next-best hypothesis under different grandparents. Whenever a new hypothesis is created, it is put onto all the agendas. Its score under different grandparenting context will be calculated, as well. Whenever a new grandparenting context (path) is found, all the existing hypotheses will be reevaluated under the new context and added to the newly created agenda. The vector *failed-dtrs* is used to skip the known failed hypotheses. Also the failure propagation function propagate-failure() is also outlined in pseudo-code. It requires that each hypothesis keeps a list of back pointers to its parent hypotheses.

6.5.5 Evaluation

To evaluate the performance of the selective unpacking algorithm, we carried out a series of empirical evaluations with the ERG and GG

```
procedure selectively-unpack-edge(edge, n) ≡
  return selectively-unpack-edge(edge, ⟨△⟩, n);

procedure selectively-unpack-edge(edge, path, n) ≡
  results ← ⟨ ⟩; i ← 0;
  do
    hypothesis ← hypothesize-edge(edge, path, i); i ← i + 1;
    if (new ← instantiate-hypothesis(hypothesis)) then
      n ← n − 1; results ← results ⊕ ⟨new⟩;
  while (hypothesis and n ≥ 1);
  return results;

procedure new-hypothesis(edge, decomposition, daughters, indices) ≡
  hypothesis ← new hypothesis(decomposition, daughters, indices);
  edge.all-hypotheses ← edge.all-hypotheses ⊕ ⟨hypothesis⟩;
  for each (path in edge.paths) do
    edge.agenda[path].insert(score-hypothesis(path, hypothesis), hypothesis);
  decomposition.indices ← decomposition.indices ∪ {indices};

procedure propagate-failure(hypothesis) ≡
  hypothesis.inst-failed ← true;
  for each (hypo-parent in hypothesis.parents) do
    propagate-failure(hypo-parent);
```

Figure 6.5: Extended selective unpacking procedure with support for arbitrary levels of grandparenting features and instantiation failure caching

```
procedure hypothesize-edge(edge , path , i) $\equiv$
  if (path not in edge.paths)
    for each (hypo in edge.all-hypotheses) do
      edge.ageanda[path].insert(score-hypothesis(path, hypo), hypo);
    paths $\leftarrow$ paths $\oplus$ $\langle$path$\rangle$;
  else if (i $<$ |edge.hypotheses(path)|)
    return edge.hypotheses(path)[i];
  if (i $=$ 0 and |paths| $=$ 1) then
    for each (decomposition in decompose-edge(edge)) do
      daughters $\leftarrow$ $\langle\,\rangle$; indices $\leftarrow$ $\langle\,\rangle$;
      for each (dedge in decomposition.rhs) do
        daughters $\leftarrow$ daughters $\oplus$ $\langle$hypothesize-edge(dedge, path $\oplus$ $\langle$edge$\rangle$, 0)$\rangle$;
        indices $\leftarrow$ indices $\oplus$ $\langle$0$\rangle$;
      new-hypothesis(edge, decomposition, daughters, indices);
  while (hypothesis $\leftarrow$ edge.agenda[path].pop() and i $\geq$ |edge.hypotheses(path)|)
    adv-indices $\leftarrow$ advance-indices(hypothesis.indices);
    for each (indices in adv-indices) do
      if (indices $\in$ hypothesis.decomposition.indices) then continue;
      daughters $\leftarrow$ $\langle\,\rangle$; failed-dtrs $\leftarrow$ $\langle\,\rangle$;
      for each (dedge in hypothesis.decomposition.rhs) each (i in indices) do
        daughter $\leftarrow$ hypothesize-edge(dedge, path $\oplus$ $\langle$edge$\rangle$, i);
        if (not daughter) then daughters $\leftarrow$ $\langle\,\rangle$; break;
        if (daughter.inst-failed) then failed-dtrs $\leftarrow$ failed-dtrs $\oplus$ $\langle$1$\rangle$;
        else failed-dtrs $\leftarrow$ failed-dtrs $\oplus$ $\langle$0$\rangle$;
        daughters $\leftarrow$ daughters $\oplus$ $\langle$daughter$\rangle$;
      if (daughters) then
        if (failed-dtrs $=$ $\vec{0}$) then
          new-hypothesis(edge, hypothesis.decomposition, daughters, indices);
        else adv-indices $\leftarrow$ adv-indices $\oplus$ $\langle$indices + failed-dtrs$\rangle$;
  if (i $<$ |edge.hypotheses(path)|) then return edge.hypotheses(path)[i];
  else return null;
```

Figure 6.5 (continued)

(Müller and Kasper, 2000; Crysmann, 2003, 2005), in combination with a modified version of the PET parser. [incr tsdb()] profiling system is used in combination with the PET parser and the aforementioned grammars to achieve fine-grained performance analyses with different configurations of the packing/unpacking algorithms. When running the ERG we used as our test set the *JH4* section of the LOGON treebank[3], which contains 1,603 items with an average sentence length of 14.6 words. The remaining LOGON treebank (of around 8,000 items) was used for training the various ME parse disambiguation models. For the experiment with the GG, we designated a 2,825 item portion of the DFKI Verb*mobil* treebank[4] for our tests, and trained ME models on the remaining 10,000 utterances. At only 7.4 words, the average sentence length is much shorter in the Verb*mobil* data.

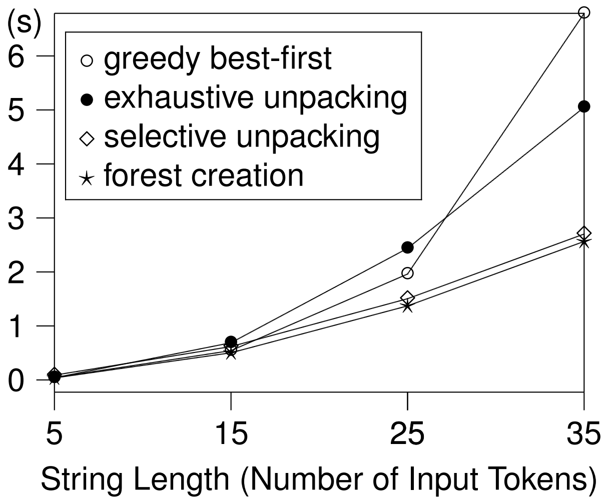


Figure 6.6: Parsing times for different configurations using the ERG, in all three cases searching for up to ten results, without the use of grandparenting.

We ran seven different configurations of the parser with different search strategies and (un-)packing mechanisms:

- Agenda driven greedy n-best parsing using the ME score without grandparenting features; no local ambiguity packing;

- Local ambiguity packing with exhaustive unpacking, without grandparenting features;

[3]The treebank consists of several booklets of edited, instructional texts on back-country activities in Norway. The data is available from the LOGON web site at '`http://www.emmtee.net`'.

[4]The data in this treebank is taken from transcribed appointment scheduling dialogs; see '`http://gg.dfki.de/`' for further information on GG and its treebank.

- Local ambiguity packing and selective unpacking for n-best parsing, with 0 through 4 levels of grandparenting (GP) features.

As a side-effect of differences in efficiency, some configurations could not complete parsing all sentences given reasonable memory constraints (which we set at a limit of 100k passive edges or 300 seconds processing time per item). The overall coverage and processing time of different configurations on *JH4* are given in Table 6.4.

Configuration	**GP**	**Coverage**	**Time (s)**
greedy best-first	0	91.6%	3889
exhaustive unpacking	0	84.5%	4673
selective unpacking	0	94.3%	2245
	1	94.3%	2529
	2	94.3%	3964
	3	94.2%	3199
	4	94.2%	3502

Table 6.4: Coverage on the `ERG` for different configurations, with fixed resource consumption limits (of 100k passive edges or 300 seconds). In all cases, up to ten 'best' results were searched, and *Coverage* shows the percentage of inputs that succeed to parse within the available resource. *Time* shows the end-to-end processing time for each batch.

The correlation between processing time and coverage is interesting. However, it makes the efficiency comparison difficult as the parser behavior is not clearly defined when the memory limit is exceeded. To circumvent this problem, in the following experiments we average only over those 1362 utterances from *JH4* that complete parsing within the resource limit in all seven configurations. Nevertheless, it must be noted that this restriction potentially reduces efficiency differences among configurations, as some of the more challenging inputs (which typically lead to the largest differences) are excluded.

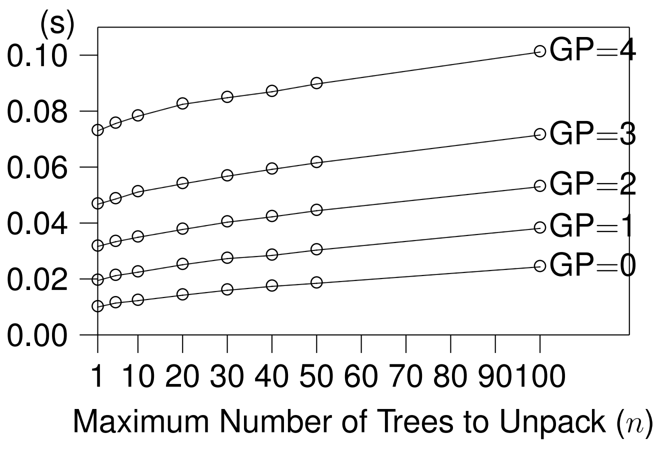


Figure 6.7: Mean times for selective unpacking of all test items for n-best parsing with the ERG, for varying n and grandparenting (GP) levels

Figure 6.6 compares the processing time of different configurations. The difference is much more significant for longer sentences (i.e., with more than 15 words). If the parser unpacks exhaustively, the time for unpacking grows with the sentence length at a quickly increasing rate. In such cases, the efficiency gain with ambiguity packing in the parsing phase is mostly lost in the unpacking phase. The graph shows that greedy best-first parsing without packing outperforms exhaustive unpacking for sentences of less than 25 words. With sentences longer than 25 words, the packing mechanism helps the parser overtake greedy best-first parsing, although the exhaustive unpacking time also grows fast.

With the selective unpacking algorithm presented earlier in the section, unpacking time is reduced, and grows only slowly as sentence length increases. Unpacking up to ten results, when contrasted with the timings for forest creation (i.e., the first parsing phase) in Figure 6.6, adds a near-negligible extra cost to the total time required for both phases. Moreover, Figure 6.7 shows that with selective unpacking, as n is increased, unpacking time grows roughly linearly for all levels of grandparenting (albeit always with an initial delay in unpacking the first result).

Table 6.6 summarizes a number of internal parser measurements using the ERG with different packing/unpacking settings. Besides the

Configuration	**Exact Match**	**Top Ten**
random choice	11.34	43.06
no grandparenting	52.52	68.38
greedy best-first	51.79	69.48
grandparenting[1]	56.83	85.33
grandparenting[2]	56.55	84.14
grandparenting[3]	56.37	84.14
grandparenting[4]	56.28	84.51

Table 6.5: Parse selection accuracy for various levels of grandparenting. The *exact match* column shows the percentage of cases in which the correct tree, according to the treebank, was ranked highest by the model; conversely, the *top ten* column indicates how often the correct tree was among the ten top-ranking results.

difference in processing time, we also see a significant difference in *"space"* between exhaustive and selective unpacking. Also, the difference in *"unifications"* and *"copies"* indicates that with our selective unpacking algorithm, these expensive operations on typed feature structures are significantly reduced.

In return for increased processing time (and marginal loss in coverage) when using grandparenting features, Table 6.5 shows some large improvements in parse selection accuracy (although the picture is less clear-cut at higher-order levels of grandparenting[5]). A balance point between efficiency and accuracy can be made according to application needs.

Finally, we compare the processing time of the selective unpacking algorithm with and without instantiation failure caching and propagation. The empirical results for `GG` are summarized in Table 6.7,

[5]The models were trained using the open-source TADM package (Malouf, 2002), using default hyper-parameters for all configurations, viz. a convergence threshold of 10^{-8}, a variance of the prior of 10^{-4}, and a frequency cut-off of 5. It is likely that further optimization of hyper-parameters for individual configurations would moderately improve model performance, especially for higher-order grandparenting levels with large numbers of features.

	Configuration	GP	Unifications (#)	Copies (#)	Space (kbyte)	Unpack (s)	Total (s)
≤ 15 words	greedy best-first	0	1845	527	2328	–	0.12
	exhaustive unpacking	0	2287	795	8907	0.01	0.12
	selective unpacking	0	1912	589	8109	0.00	0.12
		1	1913	589	8109	0.01	0.12
		2	1914	589	8109	0.01	0.12
		3	1914	589	8110	0.01	0.12
		4	1914	589	8110	0.02	0.13
> 15 words	greedy best-first	0	25233	5602	24646	–	1.66
	exhaustive unpacking	0	39095	15685	80832	0.85	1.95
	selective unpacking	0	17489	4422	33326	0.03	1.17
		1	17493	4421	33318	0.05	1.21
		2	17493	4421	33318	0.09	1.25
		3	17495	4422	33321	0.13	1.27
		4	17495	4422	33320	0.21	1.34

Table 6.6: Contrasting the efficiency of various (un-)packing settings in use with `ERG` on short (top) and medium-length (bottom) inputs; in each configuration, up to ten trees are extracted. *Unification* and *Copies* is the count of top-level FS operations, where only successful unifications require a subsequent copy (when creating a new edge). *Unpack* and *Total* are unpacking and total parse time, respectively.

showing clearly that the technique reduced unnecessary hypotheses and instantiation failures. The design philosophy of the ERG and GG differ. During the first, forest creation phase, GG suppresses a number of features (in the HPSG sense, not the ME sense) that can actually constrain the combinatorics of edges. This move makes the packed forest more compact, but it implies that unification failures will be more frequent during unpacking. In a sense, GG thus moves part of the search for globally consistent derivations into the second phase, and it is possible for the forest to contain 'result' trees that ultimately turn out to be incoherent. Dynamic programming of instantiation failures makes this approach tractable, while retaining the general breadth-first characteristic of the selective unpacking regime.

The efficient n-best unpacking algorithm allows us to selectively investigate a large number of passive edges in a packed representation without exhaustively unpacking everything. Both models (I and II) we proposed in the previous section heavily depend on the use of the selective unpacking algorithm.

6.5.6 Discussion

The n-best unpacking algorithm we have described in this section is closely related to a number of other works in the field of efficient parsing algorithms.

The closest approach to ours is Huang and Chiang (2005), where a series of increasingly efficient algorithms for unpacking n-best results from a weighted hyper-graph representing a parse forest are presented. While their final algorithm is essentially equivalent to ours, we see the main difference in that our work is specifically designed for unification-based parsing. Therefore, avoiding computationally expensive unification operations becomes the first priority in our design. Also, we specifically discussed the different localities of the features used in the parse ranking models, while Huang and Chiang (2005) did not provide any insight on this aspect. More generally, both works turn out to be reformulations of an approach originally described by Jiménez and Marzal (2000), although expressed there

Configuration	**Unifications** (#)	**Copies** (#)	**Hypotheses** (#)	**Space** (kbyte)	**Unpack** (ms)	**Total** (ms)
greedy best-first	5980	1447	–	9202	–	400
selective, no caching	5535	1523	1245	27188	70	410
selective, with cache	4915	1522	382	27176	10	350

Table 6.7: Efficiency effects of the instantiation failure caching and propagation with **GG**, without grandparenting. All statistics are averages over the 1941 items that complete within the resource bounds in all three configurations. *Unification*, *Copies*, *Unpack*, and *Total* have the same interpretation as in Table 6.6, and *Hypotheses* is the average count of hypothesized sub-trees.

only for grammars in Chomsky Normal Form.

In relation to the feature localities of ME models on packed forest, our approach considered both upward (grandparenting features) and downward (lexical n-gram features) extension. Previous work has either assumed properties that are restricted to the minimal parse fragments (i.e., sub-trees of depth one) that make up the packing representation (Geman and Johnson, 2002), or has taken a more relaxed approach by allowing non-local features but without addressing the problem of how to efficiently extract the top-ranked trees from a packed forest (Miyao and Tsujii, 2002).

Another interesting related, though different approach is that of Malouf and van Noord (2004). In their system (`Alpino` Dutch parser), a left-corner parser is used. The local ambiguities are packed into groups of so-called *"left corner spines"*, which hold larger pieces of information than the passive edges in bottom-up chart parsers. In order to recover the best readings from the packed parse forest, a beam search is used to achieve high efficiency. However, the procedure does not guarantee exact n-best results as we do in our approach. On the other hand, the beam search approach potentially has lower computational complexity, and can be practically useful for handling very long parser inputs.

When parsing with context free grammars, a (single) parse can be retrieved from a parse forest in time linear in the length of the input string (Billot and Lang, 1989). However, when parsing with a unification-based grammar and packing under features structure subsumption, the cross-product of some local ambiguities may not be globally consistent. In principle, when parsing with a pathological grammar with a high rate of failure (as we observed with `GG`), extracting a single consistent parse from the forest could take exponential time (see Lang (1994) for a discussion of this issue with respect to Indexed Grammars). In our approach, the instantiation failure caching and propagation mechanism helps reduce the effect dramatically. As future research, we think that combining a 'controlled' beam-search with variable beam width limited by the interval of score adjustment can be of great interest.

6.6 Summary

In this chapter, we presented partial parsing models for robust deep linguistic processing. Based on bottom-up chart parsing, we defined a partial parse as a path connected with passive edges on the parsing chart from the source vertex (the beginning position of the input) to the terminal vertex (the end position of the input). Several partial parse selection models have been presented and evaluated. Also the efficiency problem of partial parsing has been discussed in detail. Specifically, the selective unpacking algorithm has been incorporated, extended and thoroughly evaluated.

7 Conclusion

It is of interest to note that while some dolphins are reported to have learned English – – up to fifty words used in correct context – – no human being has been reported to have learned dolphinese.

— Carl Sagan *(1934 - 1996)*

7.1 Summary

In this dissertation, we have presented a series of robust deep processing techniques as add-on modules to existing platforms. Without any significant change to the architecture, the existing grammar resources work with the robust processing modules to achieve better performance. From lexical acquisition to MWE acquisition, to partial parsing, these techniques, though simple and standalone, together serve the same purpose to maximally utilize the linguistic knowledge encoded in the deep grammars. And the experiment results show they bring a significant performance boost to the existing deep processing systems.

Deep grammars are precious linguistic resources. Some previous robust processing proposals rely on the re-formalization of the linguistic frameworks which entails the reconstruction of the deep grammars. This has been proved to be a very difficult task. Instead, in this dissertation we have demonstrated that better robustness can be achieved by various small processing techniques. In doing so, we have released the burden of bearing robustness in mind from grammar writers, and let them focus on the description of the canonical use of the language. However, we do not deny the insufficiency of current linguistic frameworks (e.g., the incapability to model graded grammaticality in many of the current linguistic frameworks). Until a new promising framework emerges, the most practical way is to exploit the underutilized existing grammar resources.

Another important feature of the techniques proposed in this dissertation is that most of the robust components are independent of external language resources. Most of the components require the deep grammar, the treebank as manually disambiguated grammar output, and optionally a large unannotated corpus. The grammar is the core of deep linguistic processing. The treebank is usually a by-product of grammar development. It serves as preferred grammar output, and helps to build empirical prediction models to simulate the analyses licensed by the grammar. The unannotated corpora are neutral to specific linguistic frameworks, and are available for relatively low cost in different languages. They are used for discovering of insufficient robustness and the validation of the robust processing outcome. Such an *in vivo* approach enables us to adapt the techniques to various different languages, provided the three basic resources are available.

7.2 *"Trinity"* Revisited

At the beginning of the dissertation (in Section 1.2.2) I said that there are three major challenges for the deep linguistic processing, namely the problem of efficiency, specificity, and coverage. The robust processing techniques introduced in this dissertation have significantly improved the coverage of the grammar. Meanwhile, the efficiency and specificity challenges reemerge regularly. For instance, after extending the deep lexicon, higher lexical ambiguity can potentially lead to larger number of analyses and longer processing time. In partial parsing, the huge number of intermediate parsing results needs to be disambiguated accurately and efficiently. A group of new research topics are raised in those seemingly solved problems in deep linguistic processing, including but not limited to ambiguity packing and unpacking, parse filtering, parse disambiguation, etc. Predictably, these factors will continue to be closely involved in future development. A balanced treatment of the three will be crucial for better deep linguistic processing.

7.3 Future Research

Of course, there are many more open questions regarding the robustness of deep linguistic processing than those I have covered in this dissertation. Here I will list some of them that fit into our short-term future research agenda.

This dissertation focused on deep processing with hand-crafted grammars. In recent development of deep linguistic processing, the treebank-induced grammars are attracting more and more attention. Having their advantage in having faster development cycle and broader coverage, the treebank-induced grammars are deficient in being less accurate than hand-crafted grammars. An interesting direction of research would be to investigate the possibility of combining the two approaches. For instance, the treebanks created with hand-crafted grammars provide detailed analyses that are normally not available from manually annotated corpora. Therefore, a grammar induced from such treebanks can potentially bear benefits from both approaches to grammar engineering.

In this dissertation we followed the *in vivo* approach in building robust processing techniques. This is intended to keep the generality of the approach, and be as least dependent on external resources as possible. However, for a specific language, extra linguistic resources can potentially bring extra robustness to the deep processing. The idea is similar to so-called hybrid language processing where different NLP components are able to communicate with each other using a shared protocol (e.g., `RMRS`). The insufficiency of the deep grammar can be compensated by other components when necessary.

Another more general topic that urgently requires further investigation is the evaluation of deep linguistic processing systems. While shallow language processing systems are typically evaluated by comparing outputs to pre-annotated gold-standard data, it is almost impossible to annotate deep linguistic information while maintaining framework independence. Therefore, deep processing communities find it extremely difficult to compare results with each other. Any new development in cross-platform evaluation will be very helpful for deep linguistic processing in general.

7.4 Closing Words

Despite the magnificent advancement in deep linguistic processing during the past few years, we still see that the techniques remain underutilized at the time I am closing this dissertation. While the intrinsic problems of deep linguistic processing are being solved, it seems that convincing results through solid applications are still missing. The motivation of using deep analysis is not yet self-evident to the NLP application researchers. Hopefully this will change when the need for more advanced applications emerges. For instance, in recent development of machine translation, people have come to agree that syntactic analysis is necessary for higher quality translations.

At a recent workshop titled "Deep Linguistic Processing" at ACL 2007, Prague, a group of researchers from different deep processing communities came together to discuss the existing problems and future direction of DLP. This is a wonderful initial step of many to come. With strong faith and patience, I believe the era for deep linguistic processing is just ahead of us.

Bibliography

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

Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools.* Addison-Wesley, 1986.

Timothy Baldwin. Bootstrapping deep lexical resources: Resources for courses. In *Proceedings of the ACL-SIGLEX Workshop on Deep Lexical Acquisition*, pages 67–76, Michigan , USA, 2005a.

Timothy Baldwin. General-purpose lexical acquisition: procedures, questions and results. In *Proceedings of the Pacific Association for Computational Linguistics 2005*, pages 23–32, Tokyo, Japan, 2005b.

Timothy Baldwin, Emily M. Bender, Dan Flickinger, Ara Kim, and Stephan Oepen. Road-testing the English Resource Grammar over the British National Corpus. In *Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC 2004)*, Lisbon, Portugal, 2004.

Petra Barg and Markus Walther. Processing unknown words in HPSG. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics (ACL-COLING 1998)*, Montreal, Canada, 1998.

Emily Bender, Dan Flickinger, and Stephan Oepen. The Grammar Matrix: an open-source starter-kit for the rapid development of cross-linguistically consistent broad-coverage precision grammars.

In *Proceedings of the Workshop on Grammar Engineering and Evaluation at the 19th International Conference on Computational Linguistics*, pages 8–14, Taipei, Taiwan, 2002.

Sylvie Billot and Bernard Lang. The structure of shared forests in ambiguous parsing. In *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics (ACL 1989)*, pages 143–151, Vancouver, Canada, 1989.

Francis Bond, Sanae Fujita, Chikara Hashimoto, Kaname Kasahara, Shigeko Nariyama, Eric Nichols, Akira Ohtani, Takaaki Tanaka, and Shigeaki Amano. The Hinoki Treebank: a treebank for text understanding. In *Proceedings of the 1st International Joint Conference on Natural Language Processing (IJCNLP 2004)*, pages 554–562, Hainan Island, China, 2004.

Gosse Bouma and Begona Villada. Corpus-based acquisition of collocational prepositional phrases. In *Proceedings of the 12th Meeting of Computational Linguistics in the Netherlands (CLIN 2001)*, Enschede, the Netherlands, 2002.

Gosse Bouma, Gertjan van Noord, and Robert Malouf. Alpino: wide-coverage computational analysis of Dutch. In *Proceedings of the 11th Meeting of Computational Linguistics in the Netherlands (CLIN 2000)*, Tilburg, the Netherlands, 2001.

Thorsten Brants. TnT - a statistical part-of-speech tagger. In *Proceedings of the 6th Conference on Applied Natural Language Processing (ANLP 2000)*, Seattle, USA, 2000.

Eric Brill. Some advances in transformation-based part of speech tagging. In *Proceedings of the 12th National Conference on Artificial Intelligence*, volume 1, pages 722–727, Seattle, USA, 1994.

Ted Briscoe and John Carroll. Evaluating the accuracy of an unlexicalized statistical parser on the PARC DepBank. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pages 41–48, Sydney, Australia, 2006.

Ted Briscoe, John Carroll, and Rebecca Watson. The second release of the RASP system. In *Proceedings of the COLING/ACL 2006 Interactive Presentation Sessions*, pages 77–80, Sydney, Australia, 2006.

Sabine Buchholz and Erwin Marsi. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL-X)*, New York City, USA, 2006.

Lou Burnard. User reference guide for the British National Corpus. Technical report, Oxford University Computing Services, 2000.

Miriam Butt, Helge Dyvik, Tracy Holloway King, Hiroshi Masuichi, and Christian Rohrer. The Parallel Grammar Project. In *Proceedings of COLING-2002 Workshop on Grammar Engineering and Evaluation*, pages 1–7, Taipei, Taiwan, 2002.

Ulrich Callmeier. PET – a platform for experimentation with efficient HPSG processing techniques. *Journal of Natural Language Engineering*, 6(1):99–108, 2000.

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

Ulrich Callmeier, Andreas Eisele, Ulrich Schäfer, and Melanie Siegel. The DeepThought core architecture framework. In *Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC 2004)*, Lisbon, Portugal, 2004.

John Carroll, editor. *The Evaluation of Parsing Systems: Workshop at the 1st International Conference on Language Resources and Evaluation*, Granada, Spain, 1998.

John Carroll and Stephan Oepen. High efficiency realization for a wide-coverage unification grammar. In *Proceedings of the 2nd*

International Joint Conference on Natural Language Processing (IJCNLP 2005), pages 165–176, Jeju Island, Korea, 2005.

John Carroll, Anette Frank, Dekang Lin, Detlef Prescher, and Hans Uszkoreit, editors. *Proceedings of the Workshop 'Beyond PARSEVAL — Towards improved evaluation measures for parsing systems' at the 3rd International Conference on Language Resources and Evaluation*, Las Palmas de Gran Canaria, Spain, 2002.

Eugene Charniak. 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, 2000.

Ann Copestake. Robust Minimal Recursion Semantics. draft, 2006.

Ann Copestake, Dan Flickinger, Ivan Sag, and Carl Pollard. Minimal Recursion Semantics: An introduction. draft, 1999.

Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms.* MIT Press, 1990.

Berthold Crysmann. On the efficient implementation of German verb placement in HPSG. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2003)*, pages 112–116, Borovets, Bulgaria, 2003.

Berthold Crysmann. Syncretism in German: a unified approach to underspecification, indeterminacy, and likeness of case. In *Proceedings of the 12th International Conference on Head-Driven Phrase Structure Grammar (HPSG 2005)*, Lisbon, Portugal, 2005.

J. N. Darroch and D. Ratcliff. Generalized iterative scaling for log-linear models. In *Annals of Mathematical Statistics*, volume 43, pages 1470–1480. Institute of Mathematical Statistics, 1972.

Rebecca Dridan and Francis Bond. Sentence comparison using Robust Minimal Recursion Semantics and an ontology. In *Proceedings* *of the ACL Workshop on Linguistic Distances*, pages 35–42, Sydney, Australia, 2006.

Jay Earley. An efficient context-free parsing algorithm. *Communications of the Association for Computing Machinery*, 13(2):94–102, 1970.

David Elworthy. Tagset design and inflected languages. In *EACL SIGDAT workshop "From Texts to Tags: Issues in Multilingual Language Analysis"*, pages 1–10, Dublin, Ireland, 1995.

Gregor Erbach. Syntactic processing of unknown words. IWBS Report 131, IBM, Stuttgart, Germany, 1990.

Dan Flickinger. On building a more efficient grammar by exploiting types. *Natural Language Engineering*, 6(1):15–28, 2000.

Dan Flickinger. On building a more efficient grammar by exploiting types. In Stephan Oepen, Dan Flickinger, Jun'ichi Tsujii, and Hans Uszkoreit, editors, *Collaborative Language Engineering*, pages 1–17. CSLI Publications, 2002.

Frederik Fouvry. Lexicon acquisition with a large-coverage unification-based grammar. In *Proceedings of the 10th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2003)*, pages 87–90, Budapest, Hungary, 2003a.

Frederik Fouvry. *Robust processing for constraint-based grammar formalisms.* PhD thesis, Gradudate School, University of Essex, Colchester, UK, 2003b.

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

Gregory Grefenstette. The World Wide Web as a resource for example-based machine translation tasks. In *Proceedings of ASLIB, Conference on Translating and the Computer*, London, UK, 1999.

Dwayne Richard Hipp. *Design and development of spoken natural language dialog parsing systems.* PhD thesis, Department of Computer Science, Duke University, 1992.

Julia Hockenmaier. Creating a CCGbank and a wide-coverage CCG lexicon for German. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics (COLING-ACL 2006)*, pages 505–512, Sydney, Australia, 2006.

Julia Hockenmaier and Mark Steedman. CCGbank manual. Technical Report MS-CIS-05-09, Department of Computer and Information Science, University of Pennsylvania, 2005.

Liang Huang and David Chiang. Better k-best parsing. In *Proceedings of the 9th International Workshop on Parsing Technologies (IWPT 2005)*, Vancouver, Canada, 2005.

Ray Jackendoff. Twistin' the night away. *Language*, 73:534–559, 1997.

Víctor M. Jiménez and Andrés Marzal. Computation of the n best parse trees for weighted and stochastic context-free grammars. In *Proceedings of the Joint International Workshops on Advances in Pattern Recognition*, pages 183–192, London, UK, 2000.

Mark Johnson, Stuart Geman, Stephen Canon, Zhiyi Chi, and Stefan Riezler. 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, 1999.

Walter Kasper, Bernd Kiefer, Hans-Ulrich Krieger, C.J. Rupp, and Karsten Worm. Charting the depths of robust speech processing. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL 1999)*, pages 405–412, Maryland, USA, 1999.

Frank Keller. *Gradience in Grammar: Experimental and Computational Aspects of Degrees of Grammaticality.* PhD thesis, University of Edinburgh, 2000.

Frank Keller, Maria Lapata, and Olga Ourioupina. Using the Web to overcome data sparseness. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP 2002)*, pages 230–237, Philadelphia, USA, 2002.

Adam Kilgarriff and Gregory Grefenstette. Introduction to the special issue on Web as corpus. *Computational Linguistics*, 29, 2003.

Hans-Ulrich Krieger and Ulrich Schäfer. TDL - a Type Description Language for HPSG. Technical Report RR-94-37, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, 1994.

Bernard Lang. Recognition can be harder than parsing. *Computational Intelligence*, 10(4):486–494, 1994.

Irene Langkilde. Forest-based statistical sentence generation. In *Proceedings of the 1st Annual Meeting of the North American Chapter of Association for Computational Linguistics (NAACL 2000)*, Seattle, USA, 2000.

Jill Fain Lehman. *Self-extending natural language interfaces.* PhD thesis, School of Computer Science, Carnegie Mellon University, 1989.

Beth Levin. *English Verb Classes and Alterations.* University of Chicago Press, Chicago, USA, 1993.

Gordon Lyon. Syntax-directed least-errors analysis for context-free languages: a practical approach. *Communications of the ACM*, 17 (1):3–14, 1974.

Robert Malouf. A comparison of algorithms for maximum entropy parameter estimation. In *Proceedings of the 6th Conferencde on Natural Language Learning (CoNLL 2002)*, pages 49–55, Taipei, Taiwan, 2002.

Robert Malouf and Gertjan van Noord. Wide coverage parsing with stochastic attribute value grammars. In *Proceedings of the IJCNLP-04 Workshop: Beyond Shallow Analyses - Formalisms and Statistical Modeling for Deep Analyses*, Hainan Island, China, 2004.

Robert Malouf, John Carroll, and Ann Copestake. Efficient feature structure operations without compilation. *Natural Language Engineering*, 6:29–46, 2000.

Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. Efficient HPSG parsing with supertagging and CFG-filtering. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, pages 1671–1676, Hyderabad, India, 2007.

Yusuke Miyao and Jun'ichi Tsujii. Maximum entropy estimation for feature forests. In *Proceedings of the Human Language Technology Conference*, San Diego, USA, 2002.

Yusuke Miyao, Takashi Ninomiya, and Jun'ichi Tsujii. Corpus-oriented grammar development for acquiring a Head-driven Phrase Structure Grammar from the Penn Treebank. In *Proceedings of the 1st International Joint Conference on Natural Language Processing (IJCNLP 2004)*, pages 684–693, Hainan Island, China, 2004.

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

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

Stephan Oepen and John Carroll. Ambiguity packing in constraint-based parsing — practical results. In *Proceedings of the 1st Annual Meeting of the North American Chapter of Association for Computational Linguistics (NAACL 2000)*, pages 162–169, Seattle, USA, 2000.

Stephan Oepen, Kristina Toutanova, Stuart Shieber, Christopher Manning, Dan Flickinger, and Thorsten Brants. The LinGO Redwoods treebank: motivation and preliminary applications. In *Proceedings of COLING 2002: The 17th International Conference on Computational Linguistics: Project Notes*, Taipei, Taiwan, 2002.

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. 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*, Baltimore, USA, 2004.

Miles Osborne. Estimation of stochastic attribute-value grammars using an informative sample. In *The 18th International Conference on Computational Linguistics (COLING 2000)*, volume 1, pages 586–592, Saarbrücken, Germany, 2000.

Darren Pearce. A comparative evaluation of collocation extraction techniques. In *Proceedings of the 3rd International Conference on Language Resources and Evaluation (LREC 2002)*, Las Palmas, Canary Islands, Spain, 2002.

Adwait Ratnaparkhi. A maximum entropy model for part-of-speech tagging. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 1996)*, pages 133–142, Somerset, USA, 1996.

Stefan Riezler, Tracy Holloway King, Ronald M. Kaplan, Richard Crouch, John T. III Maxwell, and Mark Johnson. 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 271–278, Philadelphia, USA, 2002.

Carolyn Penstein Rosé and Alon Lavie. An efficient distribution of labor in a two stage robust interpretation process. In *Proceedings of* *the Second Conference on Empirical Methods in Natural Language Processing*, pages 26–34, Providence, USA, 1997.

Ivan Sag, Timothy Baldwin, Francis Bond, Ann Copestake, and Dan Flickinger. Multiword expressions: a pain in the neck for NLP. In *Proceedings of the 3rd International Conference on Intelligent Text Processing and Computational Linguistics (CICLING 2002)*, pages 1–15, Mexico City, Mexico, 2002.

Paul Schmidt, Axel Theofilidis, Sibylle Rieder, and Thierry Declerck. Lean formalisms, linguistic theory and applications. grammar development in ALEP. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING 1996)*, volume 1, pages 286–291, Copenhagen, Denmark, 1996.

Melanie Siegel and Emily Bender. Efficient deep processing of Japanese. In *Proceedings of the 3rd Workshop on Asian Language Resources and International Standardization at the 19th International Conference on Computational Linguistics*, Taipei, Taiwan, 2002.

Scott M. Thede and Mary Harper. Analysis of unknown lexical items using morphological and syntactic information with the TIMIT corpus. In *Proceedings of the 5th Workshop on Very Large Corpora*, pages 261–272, Beijing, China, 1997.

Hideto Tomabechi. Quasi-destructive graph unification. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics (ACL 1991)*, pages 315–322, Berkeley, USA, 1991.

Kristina Toutanova, Christoper D. Manning, Stuart M. Shieber, Dan Flickinger, and Stephan Oepen. 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, 2002.

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

Hans Uszkoreit. New chances for deep linguistic processing. In *Proceedings of the 19th international conference on computational linguistics (COLING 2002)*, Taipei, Taiwan, 2002.

Tim van de Cruys. Automatically extending the lexicon for parsing. In *Proceedings of the 11th ESSLLI Student Session*, pages 180–191, Malaga, Spain, 2006.

Gertjan van Noord. Error mining for wide-coverage grammar engineering. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL 2004)*, pages 446–453, Barcelona, Spain, 2004.

Gertjan van Noord. At Last Parsing Is Now Operational. In *Actes de la 13e Conference sur le Traitement Automatique des Langues Naturelles (TALN 2006)*, pages 20–42, Leuven, Belgium, 2006.

Gertjan van Noord, Gosse Bouma, Rob Koeling, and Mark-Jan Nederhof. Robust grammatical analysis for spoken dialogue systems. *Natural Language Engineering*, 5(1):45–93, 1999.

Aline Villavicencio. The availability of verb-particle constructions in lexical resources: how much is enough? *Journal of Computer Speech and Language Processing*, 19, 2005.

Aline Villavicencio, Valia Kordoni, Yi Zhang, Marco Idiart, and Carlos Ramisch. Validation and evaluation of automatically acquired multiword expressions for grammar engineering. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL 2007)*, pages 1034–1043, Prague, Czech, 2007.

Fuliang Weng. Handling syntactic extra-grammaticality. In *Proceedings of the 3rd International Workshop on Parsing Technologies (IWPT 1993)*, pages 319–332, Tilburg, the Netherlands and Durbuy, Belgium, 1993.

Fei Xia, Chung-Hye Han, Martha Palmer, and Aravind Joshi. Automatically extracting and comparing lexicalized grammars for different languages. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI 2001)*, pages 1321–1330, Seattle, USA, 2001.

Yi Zhang and Valia Kordoni. Automated deep lexical acquisition for robust open texts processing. In *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC 2006)*, pages 275–280, Genoa, Italy, 2006.

Yi Zhang, Aline Villavicencio, Valia Kordoni, and Marco Idiart. Automated multiword expression prediction for grammar engineering. In *Proceedings of the ACL 2006 Workshop on Multiword Expressions: Identifying and Exploiting Underlying Properties*, pages 36–44, Sydney, Australia, 2006.

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

Yi Zhang, Stephan Oepen, and John Carroll. Efficiency in unification-based N-best parsing. In *Proceedings of the 10th International Conference on Parsing Technologies (IWPT 2007)*, pages 48–59, Prague, Czech, 2007b.