Using lexical statistics to improve HPSG parsing

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

vorgelegt von
Rebecca Dridan

Dekan der Philosophischen Fakultät III: Univ.-Prof. Dr. W. Behringer

Berichterstatter: Prof. Dr. Hans Uszkoreit, Prof. Dr. Stephan Oepen

Tag der Disputation: 17. Dezember 2009

Contents

List of Figures

List of Tables

Acknowledgements

There are many people who contributed towards the completion of this thesis. I wish first to thank Hans Uszkoreit for taking me on as a student, and for his overviews and inspirations. I am also grateful to Valia Kordoni for her support, even before I landed in Germany, and for the advice and guidance she gave me from the start of my PhD. Stephan Oepen has been generous with his ideas and assistance from the very first, but I am particularly indebted to him for taking on a supervisory role, among all the other claims to his time. His feedback, advice and support have been invaluable.

Throughout my PhD I have been financially supported by the International Research Training Group (IRTG) in Language Technology and Cognitive Systems and by the PIRE program. IRTG funding not only allowed me to complete my thesis in Germany, but also enabled me to travel to conferences, summits and research meetings. As part of the PIRE program, I was a guest in the Department of Cognitive and Linguistics Sciences at Brown University and I am much obliged to Mark Johnson, Eugene Charniak and the whole BLLIP group for making me welcome and for providing me with new perspectives on my work.

The DELPH-IN community is a wonderful resource and I have benefitted greatly from the contact with other DELPH-IN members. Many, including Yi Zhang, Dan Flickinger, Peter Adolphs, Berthold Crysmann and Ulrich Schäfer have provided technical help with the various DELPH-IN tools and grammars. In addition to that, Tim Baldwin, Francis Bond and Emily Bender have been fantastic mentors. I'm also grateful to Ann Copestake and Emily Bender for hosting me on visits to Cambridge University and University of Washington respectively.

I have had many fruitful conversations with other researchers in hallways and conference centres, but I particularly want to single out James Curran and Alexander Koller for the feedback they have given me. I am also grateful to Alexander for helping me out with translation. Finally, fellow students in both IRTG and DELPH-IN have helped get me through the last few years with my sanity intact and I wish to thank them all, but especially Bart Cramer, Antske Fokkens and Jeremy Nicholson for their feedback, support and friendship.

1 Introduction

This thesis investigates the improvements that can be made by adding lexical statistics to a parser that uses hand-crafted grammars in the Head-Driven Phrase Structure Grammar (HPSG: Pollard and Sag 1994) framework. In the following chapters, we tease apart various aspects of the parser that can make use of different sorts of lexical information, and determine how particular factors affect the three-way accuracy, robustness and efficiency trade-off that is always a part of parser evaluation.

The experimental setup uses the PET parser (Callmeier 2001) together with the English Resource Grammar (ERG: Flickinger 2002), both which are part of the tools and resource collection of the DELPH-IN research initiative. This parser and grammar combination produces the sort of finely detailed syntactic and semantic analyses of natural language that have already been successfully used in applications such as information extraction and machine translation, and which will be progressively more necessary as natural language processing becomes more sophisticated. These precise and detailed analyses depend heavily on the fine-grained information encoded in the grammar lexicons, making HPSG one of the family of heavily lexicalised grammar formalisms that also includes others such as LFG (Kaplan and Bresnan 1982), LTAG (Schabes and Joshi 1991) and CCG (Steedman 2000). While the lexicon contributes much of the power of HPSG analysis, the central place the lexicon holds in the parsing process can also be a weakness, affecting the efficiency and robustness of the parser. It is these issues that we address in this thesis.

In contrast to hand-crafted grammars, treebank-derived grammars such as those used by the Collins parser (Collins 1996), are known to be robust and efficient but do not generally produce the same depth of analysis. The divide between treebank-derived and hand-crafted grammars is not a hard and fast line. Treebank-derived parsers require hand coding information at some level, both in corpus annotation and in the assumptions built into the parsers. All parsers that use hand-crafted grammars include at least some statistical component. And so, it is widely understood that the best way forward is to combine features of the different families. One theory of the best way to do this is to start with the robust shallow parsers and start trying to add more information. The reverse of this is to start with a deep precise hand-crafted core, and add robustness through the use of statistics. This thesis looks at the second approach, trying to improve the performance of a HPSG parser that uses hand-crafted grammars, by adding statistical information.

1.1 Aims and Contributions

The aim of this research is to improve the PET parser, which first requires that we define *improve*. Chapter 3 discusses parser evaluation in terms of robustness, efficiency and accuracy, with the major contribution of this section being a new granular accuracy evaluation metric, suitable for assessing the detailed semantic output the PET parser produces. Previously, all accuracy evaluation of DELPH-IN grammars used exact match as a basis, but with such fine-grained analyses, exact match is ill-suited for distinguishing between analyses which are completely incorrect, and those that vary in a small insignificant detail from the gold standard. In an application scenario, it makes more sense to assess how correct an analysis is, rather than determining whether it matches the gold standard in every feature. The Elementary Dependency Match (EDM) metric allows that more nuanced evaluation and can be parameterised to evaluate the sort of information required by an application.

Lexical information has been used to improve robustness and efficiency for parsers of varying formalisms and implementations. This thesis aims to discover whether this lexical information can be as beneficial for a hand-crafted HPSG-based grammar as it has been shown to be for groups such as Bangalore and Joshi (1999), Clark and Curran (2007b) and Matsuzaki *et al.* (2007). While the choice of supertag granularity appears to have been straightforward for other formalisms, the lexical information in HPSG comes from different parts of the grammar and so there is not an obvious candidate for *the* HPSG supertag form. Hence, one aim is to discover the ideal unit of lexical information that will be beneficial to HPSG parsing. Supertags have been defined as lexical descriptions that embody richer information than part-of-speech (POS) tags, and in particular include dependency information. In this work, we look at a range of granularities, ranging from simple part-of-speech distinctions to very fine-grained tags that incorporate a variety of lexical information. We look at the different forms possible and ask:

- Which tag forms are predictable from the available training data?
- Which tag forms are useful for different aspects of the parsing process?

The first question is particularly pertinent for the fine-grained tag forms, given that there is a limited amount of data that has been annotated with the appropriate HPSG structures. We experiment with training two taggers with different underlying statistical models to see which gets the better standalone tag accuracy using the available gold standard data and find that the TnT POS tagger (Brants 2000a) achieved higher tag accuracy for fine-grained tags for this amount of data than the C&C supertagger (Clark and Curran 2007b). However we find that when using the underlying probability distribution to guide tag assignment, rather than just evaluating the most likely tag, in most cases the C&C supertagger gives better results. We also experiment with adding additional data and find that a small amount of domain-specific gold standard data is very effective in increasing tag accuracy, but a more cost-effective method of obtaining extra training data is to use the uncorrected output of the parser. This reinforces the indications given by Prins and van Noord (2003), but is a more reliable result, since we evaluate against gold standard annotations, rather than against parser output. The question remaining from this portion of the research is whether standalone tag accuracy gives a good prediction as to the effect of the lexical information on the parsing process.

The answer to that question is no, not really. The second part of evaluating these different tag forms is to assess the usefulness of the information they convey to the parsing process. We look at three different uses of the lexical information and find that in most cases the tags as predicted by the C&C supertagger yield greater improvements. Even in situations where earlier results would suggest TnT has an advantage, such as when only tagging with the single most likely tag, the tags from C&C lead to greater parser robustness. In part this was due to the kinds of tags that TnT tended to get wrong, a trend that could be detected by thorough error analysis of standalone tag accuracy. But the errors in closed class tags would not be enough to explain the difference in performance. Nor does sentence accuracy appear to favour the C&C supertagger in these situations. It appears that the C&C supertagger is just better at predicting tag *sequences* that are acceptable to the parser. While this does not always mean that the analysis is completely correct, it means that the parser is able to find an analysis that is mostly correct, where the tags from the TnT tagger lead to no analysis at all. This sort of trend in tag prediction is difficult, if not impossible to predict from tag accuracy alone, since it involves complex interactions that are encoded into the grammar, and can only really be detected by running the parser.

Regarding the effective usefulness of the lexical information, we look at three different aspects of the parsing process, aiming to discover, for each aspect, *which* information helps and *how effective* is that information. First we look at increasing parser robustness by using lexical statistics to predict information about unknown words in the input. For this task, we find that the current default unknown word handling method of using Penn Treebank POS tags gives the best compromise between robustness, efficiency and accuracy. They provide an appropriate amount of information at a higher level of accuracy than similarly detailed tags that have been trained on HPSG structures, primarily because the tagger model for the Penn Treebank tags was trained on a greater amount of training data. However, we did find that when precision has a much greater priority than robustness or efficiency, supertags formed from HPSG lexical types from the grammar could be appropriate. While not bringing the same coverage as Penn Treebank tags, they bring significant advantage over using no unknown word handling and greater precision than the Penn Treebank tags for the sentences that do manage to parse. In the current parser implementation, having many fine-grained tags available for unknown word handling can make parsing less efficient, but that is something that could be rectified with a minor change to the parser.

When we focus on increasing parser efficiency, by reducing lexical ambiguity (the method often referred to in the literature as simply *supertagging*), our oracle experiments show that the potential improvement from using lexical information in this fashion is much less than in other formalisms. Rather than speed increases of 30 or 300 times, as others have reported, the best possible speed-up was found to be around twelve times the unrestricted parsing speed, when using very fine-grained gold standard tags. When using tags predicted by the best performing tagging models we had, we found that it was possible to double parsing speed with no loss of accuracy or robustness, but that any further speed increases came with an associated drop in coverage. The very simplest of the 'supertags' was found to be the most effective up to double the parsing speed, although at higher levels of restriction (and hence higher speeds), the tags based on lexical types again gave the best trade-off between coverage and speed.

The final aspect of parsing where lexical statistics have been shown to be useful is in parse ranking. In this work, we carry out a preliminary exploration of the data available in order to give an indication of what information would be the most beneficial to add to the statistical parse ranking model. The most interesting conclusion from this investigation is that, unlike in the other experiments, predicting morphology information seems likely to yield parsing improvements, in terms of disambiguating between likely analyses.

1.2 Thesis Outline

This thesis is laid out in the following manner: the first two chapters provide the necessary background of the framework, tools and methods used (Chapter 2) and of parser evaluation (Chapter 3); then Chapter 4 addresses the question of tag predictability; while the next three chapters look at the usefulness of the different tags in terms of parser robustness (Chapter 5), efficiency (Chapter 6) and accuracy (Chapter 7). Finally, Chapter 8 summarises the main results and discusses the implications of these results for future research. Below we summarise each chapter:

Chapter 2: Background In this chapter, we explain relevant characteristics of the HPSG formalism and of the PET parser that is used for experimentation. We situate the parser and grammar combination in the broader parser space, and discuss the respective advantages and disadvantages of deep hand-crafted grammars, as opposed to other systems that are derived from treebanks or that provide a less detailed analysis. We find that many disadvantages of the deep hand-crafted system come from the same sources as the advantages this combination yields. Rather than eliminate the disadvantages by eliminating the advantages, we look to combine advantages from different systems. An overview of hybrid processing examines methods that have worked in other situations, and we then focus on *supertagging* as one method particularly appropriate for a lexicalised formalism such as HPSG. Finally we describe our method for dealing with tokenisation mis-match, a problem that commonly occurs when integrating information from disparate sources.

Chapter 3: Parser Evaluation In order to measure parser improvement, we need to define 'improve'. This chapter discusses the history of parser evaluation, outlining the issues involved in evaluating robustness and efficiency, and then gives a detailed overview of parser accuracy evaluation metrics. Building on the conclusions drawn from this overview, we then define a new granular evaluation metric, Elementary Dependency Match (EDM), that is suitable for evaluating the detailed semantic information that is produced by the PET parser.

Chapter 4: The Supertagger We define eight tag forms based on the distinctions made in the English Resource Grammar, ranging from a simple 13-tag set of part-of-speech tags that would not generally be considered supertags, up to very fine-grained descriptions that include information about morphology, subcategorisation and selectional preference of prepositions. We first attempt to train two different taggers, one Hidden Markov Model-based and the other based on a Maximum Entropy Markov Model, on the 157,920 tokens of gold standard parser data available for this grammar. We then try and supplement this data, first by using a small amount of additional domain-specific gold standard data, and then with automatically annotated data. Results from these experiments indicate that the HMM-based tagger achieves better results on less training data, when evaluating only the single best tag predicted by the tagger. However, if sufficient in-domain training data is available, or when the method of assigning tags makes use of the underlying probability distribution model learnt by the tagger, the Maximum Entropy-based tagger has superior performance.

Chapter 5: Unknown Word Handling This chapter looks at using lexical statistics to predict information about an unknown word, in order to boost parser robustness. We use different granularities of tag forms that are based on the distinctions made within the grammar, and compare them to using no external information and to using Penn Treebank-style part-of-speech tags predicted by a tagger trained on one million words. While robustness is the main focus of these experiments, we also evaluate the effects on efficiency and accuracy when using each tag form.

Chapter 6: Restricting Parser Search Space This chapter considers the impact of lexical ambiguity on parser efficiency. This has been the focus of previous work involving supertags, and following that work, we use the predicted tags of different forms to restrict the lexical items that are considered in parsing. We first carry out an oracle experiment with each tag type, in order to determine the upper bound on performance possible by using this technique. Then, three different methods of lexical restriction are used: first, we allow only lexical items compatible with the single top tag predicted for each token; then we try allowing multiple tags, depending on the probabilities assigned by the taggers to the tags; finally we selectively restrict particular tokens of the input, depending on the probability assigned by the tagger to the top tag.

Chapter 7: Parse Ranking We discuss the manner in which lexical statistics could be added to the statistical parse ranking model, and how this application of lexical statistics differs from the previous experiments. Preliminary experiments are carried out to investigate what information is available in the statistics we can learn from the available training data, in order to give some indication of the effect this data could have when properly integrated into the statistical parse ranking model.

Chapter 8: Conclusion This chapter reiterates the main results that we have seen and describes future refinements and extensions of this work that could lead to further parser improvements.

2 Background

This chapter describes the PET parser and its associated DELPH-IN grammars, situating it among other parsers and grammars currently in the field. We outline some of the advantages and disadvantages of using a deep, hand-crafted grammar and, in Section 2.2, discuss some previous work on combining the advantages of deep and shallow systems. Then, in Section 2.3, we focus on *supertagging* as one specific method of using lexical statistics to improve deep parsing, examining how people have used it, and how effective it has been. Finally Section 2.4 describes some initial work that was required to be able to combine information from different sources in our parser.

2.1 HPSG Parsing

The research in this thesis uses deep, hand-crafted grammars in the Head-Driven Phrase Structure Grammar framework. These grammars are very different to the more well-known Collins (Collins 1996, Collins and Koo 2005) or Charniak (Charniak 2000, Charniak and Johnson 2005) probabilistic context free grammars (PCFGs) in what they output, how they are created and how they are used in parsing. This section describes the formalism, parser and grammars and compares these to other current systems.

2.1.1 Head-Driven Phrase Structure Grammar

Head-driven Phrase Structure Grammar (HPSG: Pollard and Sag 1994) is a unification-based formal theory of language which models syntax and semantics together in typed feature structures. In this formalism, types are associated with features which in turn take other typed feature structures as their values. These values can be atomic, or, frequently, other complex feature structures, and the types are arranged in a type hierarchy, allowing for abstraction and generalisation. The building blocks of an HPSG analysis are instances of the *word* and *phrase* types, which are subtypes of *sign*. Each sign contains detailed information about the text span it represents, with the information necessary for constraining the use of the sign encapsulated in the SYNSEM feature of the sign. A full analysis is constructed by combining these signs using generic rules that conform to a few key principles. The Head Feature Principle states that a headed phrase and its head daughter share the same HEAD value. The Valence Principle specifies that each feature of VAL in a headed phrase is the same as that of its head daughter, unless a rule states otherwise, for instance by discharging a specific VAL element. There are also semantic principles playing analogous roles regarding the semantic content of the sign. The Semantic Inheritance Principle controls which semantic features are passed from the head daughter up to the mother, while the Semantic Composition Principle states that the RESTR feature of the mother is the composition of the RESTR features of each daughter. This RESTR feature contains the semantic restrictions each sub expression imposes on the world, if the expression is to be true.

Somewhat simplified, the rules and principles specify how the signs unify but the information that constrains which signs can be unified with each other is encoded into the signs themselves, and ultimately comes from the lexical entries. Figure 2.1(b) shows an example of how the words in Figure 2.1(a) can be combined to form an analysis. We see here that the lexical entry for *love* contains the information that *love* is a transitive verb, requiring a subject and a complement. When the *third_sing_verb* morphological rule is applied to this lexical entry, a sign is created for the word *loves* that includes this information, and further specifies that the subject must have third person singular agreement. The information in the signs for the words *Kim* and *Sandy* comes directly from the lexical entries.

The head-complement rule creates a headed phrase from a head daughter and a non-head daughter, specifying that the COMPS value of the headed phrase is the COMPS value of the head daughter minus the SYNSEM of the non-head daughter. In effect, applying the head-complement rule to two signs allows the non-head daughter to 'cancel out' (or fulfil) one element of the head daughter's COMPS list. This is the rule that combines the words *loves* and *Sandy*. The *synsem* that is the first (and only) element of the COMPS value in *loves* is unified with the SYNSEM value of *Sandy*. If any features in the two synsems were incompatible, the unification would fail. Since the synsem from the COMPS value is completely underspecified in this simplified case, there is no incompatibility.

The head-subject rule also creates a headed phrase from a head daughter and a non-head daughter. In this case, the rule further specifies that the SUBJ value of the headed phrase is the SUBJ of the head daughter minus the SYNSEM of the non-head daughter. When this rule is applied to *Kim* and the phrase constructed from *loves* and *Sandy*, we can see how unification of the SYNSEM value from *Kim* and that specified in the SUBJ value of the word *loves* (and passed up via the Valence Principle) enforces agreement between the subject and the verb.

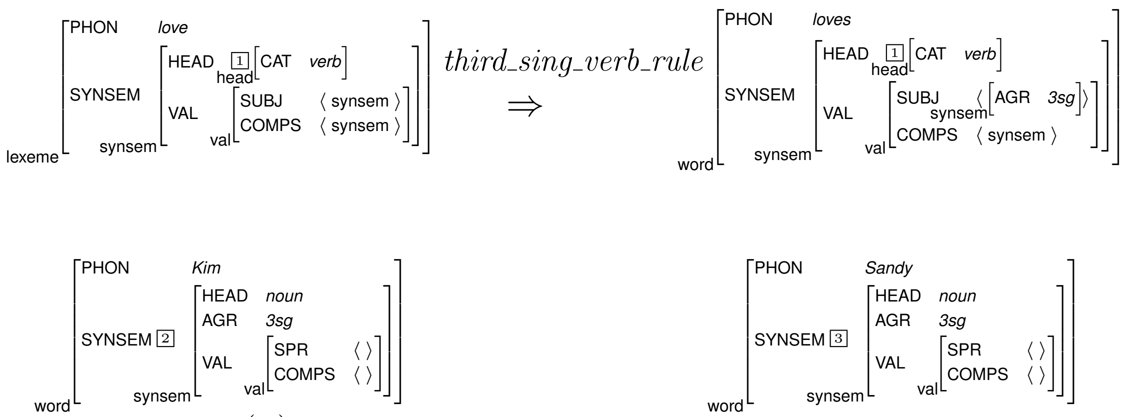
This example is highly simplified, leaving out, among other things, all semantic detail, but a full analysis works on the same general principle: the rules outline general patterns, defining which features must unify in all cases and the lexical entries specify constraints related to the lexical item. This makes the lexicon a central part of an HPSG grammar.

2.1.2 The PET Parser

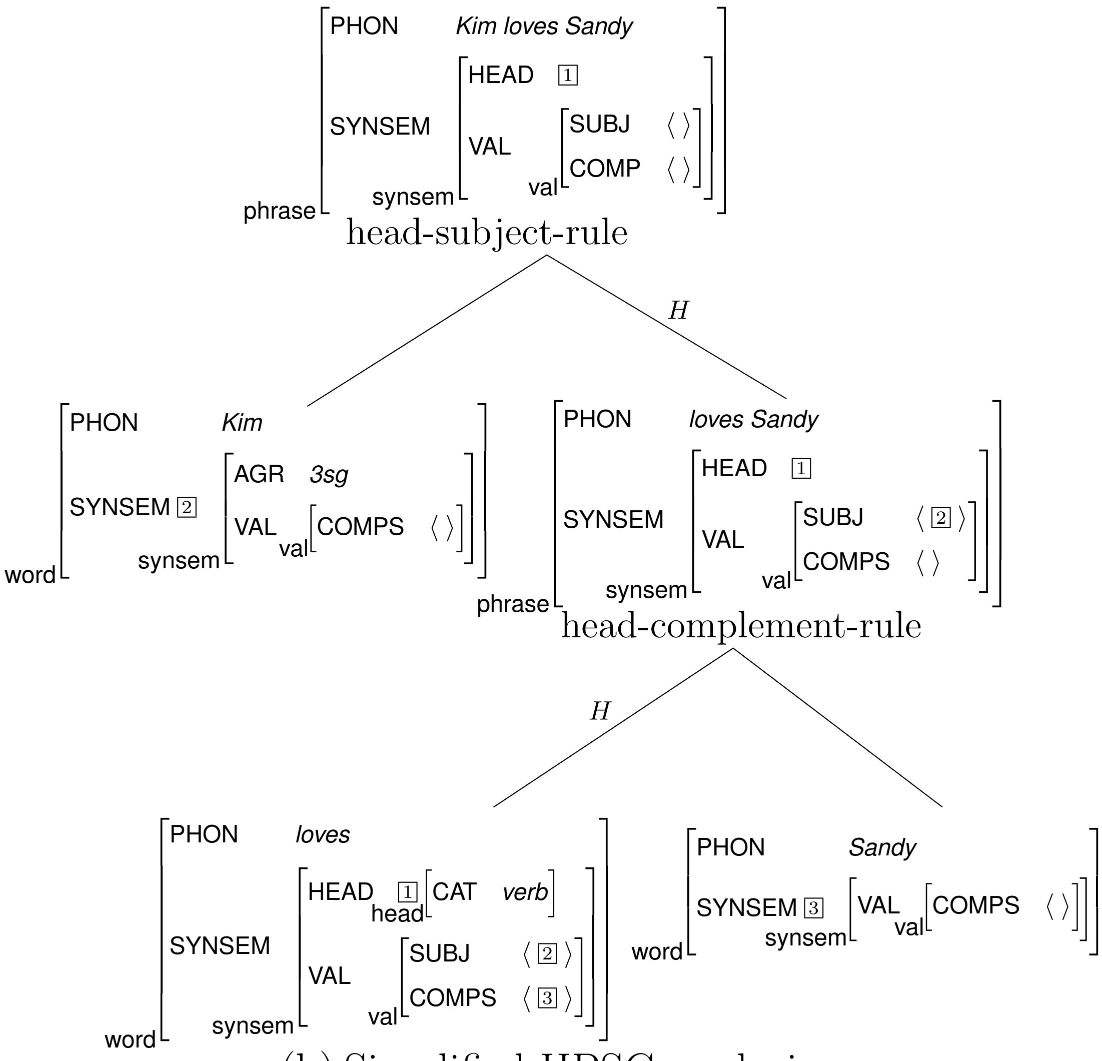
The PET parser (Callmeier 2000) is an efficient, unification-based HPSG parser that was originally developed as part of a platform for testing various unification algorithms. It has been continually enhanced and extended since that time, with the addition of various input and output formats, subsumption-based packing for increased efficiency (Oepen and Carroll 2000) and statistical parse ranking (Zhang *et al.* 2007b). The parser is part of a set of tools and grammars developed and maintained by the DELPH-IN[1] community, an international consortium of researchers from universities and institutes across the world who share an aim of using deep linguistic processing to get at the meaning of human language. DELPH-IN grammars are bidirectional grammars, used for parsing and generation, and all use Minimal Recursion Semantics (MRS: Copestake *et al.* 2005) as a common semantic representation. DELPH-IN maintains a commitment to multilinguality and has, largely through the LinGO Grammar Matrix project (Bender *et al.* 2002), developed many standard analyses for common cross-language phenomena. There are currently large broad-coverage DELPH-IN grammars for English (ERG: Flickinger 2002), Japanese (Jacy: Siegel and Bender 2002) and German (GG: Müller and Kasper 2000, Crysmann 2003), and smaller grammars for other languages such as Modern Greek (Kordoni and Neu 2004), Spanish (Marimon *et al.* 2007), French (Tseng 2003), Korean (Kim and Yang 2004) and Wambaya (Bender 2008).

The basic algorithm that PET uses to process text operates on a lattice of input tokens, externally generated, or produced from raw text using

[1]http://www.delph-in.net/


PHON love
SYNSEM HEAD 1 CAT verb head
VAL SUBJ ⟨ synsem ⟩ COMPS ⟨ synsem ⟩ val
synsem
lexeme
third_sing_verb_rule ⇒
PHON loves
SYNSEM HEAD 1 CAT verb head
VAL SUBJ ⟨ AGR 3sg synsem ⟩ COMPS ⟨ synsem ⟩ val
synsem
word
PHON Kim
SYNSEM 2 HEAD noun AGR 3sg VAL SPR ⟨ ⟩ COMPS ⟨ ⟩ val synsem
word
PHON Sandy
SYNSEM 3 HEAD noun AGR 3sg VAL SPR ⟨ ⟩ COMPS ⟨ ⟩ val synsem
word

(a) Lexical items used in the analysis


PHON Kim loves Sandy
SYNSEM HEAD 1 VAL SUBJ ⟨ ⟩ COMP ⟨ ⟩ val synsem
phrase
head-subject-rule
H
PHON Kim
SYNSEM 2 AGR 3sg VAL COMPS ⟨ ⟩ val synsem
word
PHON loves Sandy
SYNSEM HEAD 1 VAL SUBJ ⟨ 2 ⟩ COMPS ⟨ ⟩ val synsem
phrase
head-complement-rule
H
PHON loves
SYNSEM HEAD 1 CAT verb head VAL SUBJ ⟨ 2 ⟩ COMPS ⟨ 3 ⟩ val synsem
word
PHON Sandy
SYNSEM 3 VAL COMPS ⟨ ⟩ val synsem
word

(b) Simplified HPSG analysis

Figure 2.1: Lexical items and the (simplified) HPSG analysis for *Kim loves Sandy*

the built-in preprocessor. The orthographic sub-rules associated with a set of morphological rules are applied to each input token to produce possible base lemmas. These potential lemmas are looked up in the lexicon and a lexical item is added to the parsing chart for each matching lexical entry in the lexicon. The morphological rule related to the orthographic rule that produced the appropriate lemma is then applied to the relevant lexical item to produce chart items that represent word signs. The morphological rule application may be not be successful, and in that case no chart item would be produced. For example, the orthographic sub-rule associated with a morphological rule for third person singular finite verbs applied to *ups* could hypothesise the lemma *up*, but applying the rule to the lexical items returned from the lexicon would fail for all non-verb *up* items. An agenda-based chart parsing algorithm then operates over the valid word items, unifying them according to the grammar rules to produce spanning edges with a phrasal type. At each unification the chart is checked to see whether the most recent edge created subsumes or is subsumed by a pre-existing edge. In this case the edges are packed appropriately, as described by Oepen and Carroll (2000), so one edge represents all other edges that cover the same span and are more specific than the representative edge. A parse is successful when there is an edge that spans the entire input *and* has a type designated as a valid root condition. If the parser has been run to produce a limited number of analyses, the required analyses are unpacked from the chart in an order dictated by the statistical model released with the grammar, a process called selective unpacking (Carroll and Oepen 2005; Zhang *et al.* 2007b). The entire packed forest is created during parsing, and the selective unpacking is guaranteed to find the best analyses according to the statistical model.

One difference to many other parsers is the fact that the semantic analysis is built up with the syntactic analysis, rather than as a post-processing step. This fact affects the distinctions made within the grammar, since the semantic details are an integral part of the lexical entries. The semantic construction is guided by features of the lexical type which can, for example, force co-indexation of the correct semantic indices in raising verbs. As stated above, the common semantic representation for DELPH-IN grammars is Minimal Recursion Semantics (MRS). MRS is a flat semantic formalism that represents semantics with a bag of *elementary predications* and a list of scopal constraints. An elementary predication can be directly related to words in the text, or can have a

$$\begin{bmatrix} \textsc{text} & \text{He persuaded Kim to leave.} \\ \textsc{ltop} & h1 \\ \textsc{index} & e2 \\ \textsc{rels} & \left\{ \begin{array}{ll} \begin{bmatrix} \multicolumn{2}{l}{\texttt{pron\_rel<0:2>}} \\ \textsc{lbl} & h3 \\ \textsc{arg0} & x4 \end{bmatrix} & \begin{bmatrix} \multicolumn{2}{l}{\texttt{pronoun\_q\_rel<0:2>}} \\ \textsc{lbl} & h5 \\ \textsc{arg0} & x4 \\ \textsc{rstr} & h6 \\ \textsc{body} & h7 \end{bmatrix} \\ \begin{bmatrix} \multicolumn{2}{l}{\texttt{\_persuade\_v\_of\_rel<3:12>}} \\ \textsc{lbl} & h8 \\ \textsc{arg0} & e2 \\ \textsc{arg1} & x4 \\ \textsc{arg2} & x10 \\ \textsc{arg3} & h9 \end{bmatrix} & \begin{bmatrix} \multicolumn{2}{l}{\texttt{named\_rel<13:16>}} \\ \textsc{lbl} & h14 \\ \textsc{arg0} & x10 \\ \textsc{carg} & Kim \end{bmatrix} \\ \begin{bmatrix} \multicolumn{2}{l}{\texttt{proper\_q\_rel<13:16>}} \\ \textsc{lbl} & h11 \\ \textsc{arg0} & x10 \\ \textsc{rstr} & h12 \\ \textsc{body} & h13 \end{bmatrix} & \begin{bmatrix} \multicolumn{2}{l}{\texttt{\_leave\_v\_1\_rel<20:26>}} \\ \textsc{lbl} & h15 \\ \textsc{arg0} & e16 \\ \textsc{arg1} & x10 \\ \textsc{arg2} & p17 \end{bmatrix} \end{array} \right\} \\ \textsc{hcons} & \{h6 \text{ qeq } h3, h9 \text{ qeq } h15, h12 \text{ qeq } h14\} \end{bmatrix}$$

Figure 2.2: MRS for *He persuaded Kim to leave*

grammatical function, such as a quantifier. Each elementary predication has a relation name, a label and an index (designated ARG0). Arguments of an elementary predicate are represented by ARGn features, which reference the relation that fulfils the argument role. Figure 2.2 shows the MRS representing the semantic analysis of *He persuaded Kim to leave.* Here we see six elementary predicates, four with text referents and two quantifier predications. The ARG1, ARG2 and ARG3 roles of the verbal predicates describe the predicate argument relations and demonstrate co-indexation between the ARG2 of *persuade* and the ARG1 of *leave.*

2.1.3 Relationship of PET to Other Parsers

This section discusses other parsers currently in active development and use in the NLP community, and how they compare to the PET parser.

For this discussion *the PET parser* actually refers to the combination of the unification-based parser and an appropriate HPSG grammar to better compare with other parsers that don't have the same modularity between parser and grammar.

One way that parsers can be characterised is according to the depth of their analysis, where depth refers to the amount of information in the analysis, particularly the level of semantic detail. PET belongs to a family of parsers that produce deep linguistic analysis, where *deep* in this context means that the parser is capable of identifying long-distance dependencies and produces at least predicate-argument level semantics. Other deep parsers include the C&C parser (Clark and Curran 2007b) implementing the Combinatory Categorical Grammar formalism (CCG: Steedman 2000), Lexical Functional Grammar (LFG: Dalrymple 2001) parsers such as those from PARC (XLE: Riezler *et al.* 2002) and DCU (Cahill *et al.* 2004) and other HPSG-based parsers like ENJU (Miyao and Tsujii 2005) and ALPINO (van Noord and Malouf 2004). At the other end of this spectrum are PCFG parsers (Collins 1996, Charniak and Johnson 2005, inter alios) that produce basic phrase structure trees. Dependency parsers such as the MSTParser (McDonald *et al.* 2005) or MALTParser (Nivre *et al.* 2004) are somewhere in the middle, producing word-to-word surface-based dependencies and making some effort towards recovering long distance dependencies.

Another differentiating characteristic of parsers is how their grammars are built. The DELPH-IN grammars used by the PET parser are all hand-crafted grammars that have benefitted from many years of grammar engineering by linguists and hence consist of very detailed lexical and syntactic information. Other fully hand-crafted grammars are ALPINO, PARC XLE and RASP (Briscoe *et al.* 2006). The opposite of this situation is when the grammar is learnt from the annotations of a treebank. The most common treebank in use (for English) is the Penn Treebank (Marcus *et al.* 1993), which was annotated with phrase structure trees, and later enhanced with traces, null elements and function tags (Marcus *et al.* 1994). Although the enhancements were carried out because many users wanted a richer linguistic annotation, most parsers only use the phrase structure annotation. The Collins and Charniak parsers both fit this category, although some versions of the Collins parser extract specific information from the enhancements. In some cases, a treebank has to be converted to the right format for the parser. Both MALTParser and MSTParser used head finding rules from Yamada and Matsumoto

(2003) to convert the Penn Treebank phrase structure annotation to an appropriate dependency format for use in training their English parsers. The C&C parser was trained on CCGBank (Hockenmaier and Steedman 2007), which was also created by converting the Penn Treebank, although this conversion required a lot more pre-processing and validation than the dependency conversions and made full use of the enhanced annotations. The ENJU HPSG parser falls somewhere in between the hand crafted and treebank-derived parsers: the grammar rules were all hand written, but the lexical details were learnt from a pre-processed version of the Penn Treebank (Miyao *et al.* 2004). Since HPSG is a lexicalist formalism, this means that the majority of the grammar was derived from the treebank, but the derivation process and the rule creation used a lot of linguistic knowledge.

Parsers that use treebank-derived grammars have often been called statistical parsers but this can be a misleading term. All current parsers use statistics at some point in the parsing process since, at the very least, a parser that is to be used in applications needs to be able to select the most likely analysis from all those that are syntactically possible. The PET HPSG parser is one of several (e.g., Clark and Curran 2007b, Miyao and Tsujii 2008, van Noord and Malouf 2004) that build the entire parse forest and then use a statistical model to determine the most likely parse from the forest. In the case of PCFG parsers, it is computationally intractable to build the entire parse forest because of the relatively unconstrained nature of these grammars. However, by assuming independence of subtrees in the analysis, PCFG parsers (e.g., Charniak and Johnson 2005, Collins 1996) are able to use a simple generative statistical model to guide the parsing. This allows them to introduce statistics early in the parsing process, assigning probabilities to each rule application and discarding less likely rules at each point. As Abney (1997) explains, attribute-value grammars like HPSG that use re-entrancy (co-indexation between features) cannot legitimately make this independence assumption and so they can not use the same simple statistical model.

Common wisdom was that a grammar was either shallow and learnt from a treebank, or deep and handcrafted, but as described above and summarised graphically in Figure 2.3, modern parsers vary across both spectra and there are advantages and disadvantages to all configurations. Treebanks bring many advantages like relatively rapid grammar development, broad coverage (given a large treebank), and an easily accessed source of statistics. In essence, the linguistic knowledge in a treebank-

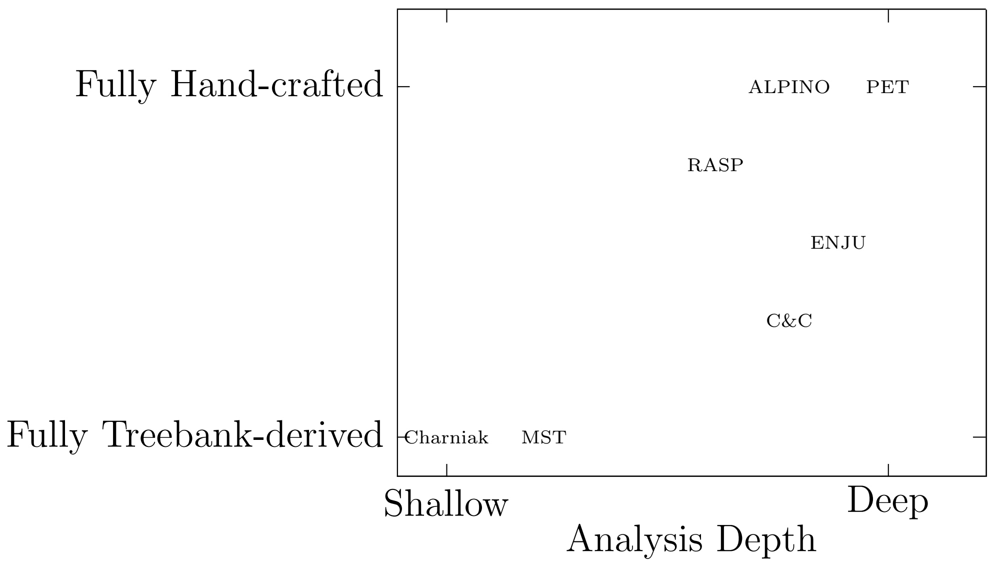


Figure 2.3: Graphical representation of parser relationships

derived grammar is the knowledge that was encoded in the treebank annotations. However, a treebank can be only as informative as its annotation scheme, and annotating a treebank is a long and expensive process. The Redwoods treebanks (Oepen *et al.* 2004b) demonstrate one method of overcoming these disadvantages, being dynamic treebanks created *from* the grammar. Since these treebanks are grammar-dependent, they don't bring the same rapid development advantages, but they do provide the statistics required to build a parse ranking model. Section 3.2.1 will go into more details regarding the creation and use of these treebanks.

One significant advantage that hand-crafted grammars have over those derived from treebanks is generalisation. Particularly for semantic analysis, it is useful to know that *seen* and *saw* have the same base form and to relate them to other transitive verbs that have similar usage patterns. While it would be possible to have this information in a treebank, most grammar-independent treebanks do not contain this information. Another problem for treebank-derived grammars is that they are heavily influenced by the domain and genre of the treebank they were created from. Given the central place of the Penn Treebank in English parsing, we now have parsers that are very good at parsing newspaper text, but significantly worse at other domains. Recent research into domain adaptation (e.g. Hara *et al.* 2007, McClosky *et al.* 2006b) has examined techniques for overcoming this problem.

Examining the advantages of deep versus shallow analysis, it is difficult to see a disadvantage to deep, detailed, accurate information, all else being equal. The fact is, though, that not all else is necessarily equal. The fact that many of the popular parsers produce a relatively impoverished level of analysis is primarily due to engineering issues. In the first place, hand annotating a treebank with a full HPSG structure is practically impossible. The simple sentence *Kim loves Sandy* produces an analysis with over 900 values that no person would be able to consistently reproduce with any accuracy. Annotation tools could provide some assistance, but not enough to make constructing a large treebank by hand practical. Strategies such as those in the HPSG extraction from Penn Treebank described by Miyao *et al.* (2004) provide one option but still depend on possibly error-prone heuristics. Another issue related to deep analysis is the sheer number of distinctions being made, and the effect this has on any statistical processing. Around seventy node labels (pre-terminal POS tags and other non-terminals such as VP and S) are used in the Penn Treebank, and at this level of distinction, an annotated corpus of one million words is enough to build up a reasonable statistical model. In a large HPSG grammar, where even the set of pre-terminals (lexical types) can number around 600–800 types, data scarcity becomes a much greater problem. The statistical models are also more complicated because they can not depend on the subtree independence assumption.

One possible disadvantage sometimes assigned to deep analysis is that of unnecessary distinctions and hence spurious ambiguity. While this can indeed be an issue, it is more an issue of how the analysis is used — it is, after all, often much easier to ignore unnecessary distinctions than to add missing information after the fact. Another accusation commonly levelled against deep processing is lack of speed. While this has been, and can be, an issue with deep parsers, recent work, particularly that involving the C&C parser (Clark and Curran 2007b) or the ENJU parser (Matsuzaki *et al.* 2007), has shown that slow processing is not an inherent factor in deep parsing.

The PET parser produces deep detailed syntactic and semantic analyses but, like other parsers, it has some factors that limit its use in applications. In the case of PET, the main factor is robustness: the ability to produce an analysis regardless of circumstances. Baldwin *et al.* (2005) describe some of the reasons for parse failure which include ungrammatical input, extra-grammatical input (i.e., ungrammatical according to the grammar of the parser, but not to a native speaker) and resource exhaustion. Each of these causes of failure requires a different strategy to overcome. In the case of ungrammatical input, there are occasions where it is preferable for the parser to fail, such as a grammar checking application or any application that requires precision before recall. Other situations might benefit from a 'best guess' of the intended meaning by the parser, optionally marked as ungrammatical. Work from Bender *et al.* (2004) and Crysmann *et al.* (2008) describe one way to do this using mal-rules, but this is limited to parsing predictable errors. A broader coverage solution would require the relaxing of constraints in the grammar. For extra-grammatical input, the obvious strategy is to improve the grammar and there has been research into methods of assisting the grammar writer in this aim (van Noord 2004, Zhang and Kordoni 2006, Goodman and Bond 2009). However, even with complete constructional coverage, ever-changing language means it is not practical to assume that a grammar will contain every possible word in use and hence other strategies to handle unknown words are required.

Resource exhaustion can refer to overstepping memory, space or time limits, whether hard limitations of the hardware or limits applied to ensure reasonable parse times. The obvious solution to parse failures caused by resource limits (other than increasing the limits, which is not always possible) is to make the parser more efficient. The addition of subsumption-based packing (Oepen and Carroll 2000) and selective unpacking (Zhang *et al.* 2007b) strategies to the PET parser have increased parser efficiency significantly, but there is always room here for improvement.

Many of the robustness issues the PET parser has can be attributed to specific features of the parser that differentiate it from other parsers: the inherent idea of grammaticality encoded in part by hard constraints, the detailed syntactic and semantic nature of the analyses and, relatedly, the detailed lexicon with its precise distinctions. Rather than eliminate the issues by eliminating the advantages of the parser, a better solution comes from combining the advantages of different parsers and other language processing tools. The next section outlines some of the previous work in so-called hybrid processing.

2.2 Hybrid Processing

Hybrid processing can refer to any number of ways of combining the information produced by different natural language processing tools. As discussed in the last section, often the features that give a parser its advantages are also responsible for certain disadvantages. By combining tools with different advantages and disadvantages, we can produce a better overall system. This can mean using both different methods of analysis, but also different sources of information. For instance, statistical information about word collocation frequencies and semantic information about word meanings can both have some influence on how a sentence is interpreted.

One issue of hybrid processing is how to combine the information from different tools. The simplest method is that of fall-back: tools are ordered according to the desirability of their output (generally, deeper is better) and the output of the best tool to produce an analysis within a given time is used. Another option is to merge the output of complementary processing tools, adding depth to the final analysis. Finally, it is possible to integrate information from different sources within, say, a parser. The Heart of Gold (Callmeier *et al.* 2004; Schäfer 2006) is a system that uses all three techniques: integrating information by annotating parser input with part-of-speech (POS) tags, falling back to the best parser that produced an output and then merging that parser output with the output of a named entity recogniser.

Integrating is the most relevant strategy to examine when the goal is to improve HPSG parsing, but it has certain complications, the most frequent being mismatch of assumptions that the different tools make. This can be in the form of different distinctions, such as in named entity classifications or in word categorisations, perhaps emphasising syntactic over semantic distinctions. Another frequent mismatch is tokenisation. While this may appear to be a trivial issue, disagreements over the definitions of the fundamental building blocks of an analysis can cause serious problems when attempting to integrate different forms of information.

Most previous work on hybrid processing by integration adds extra information to a parser, often by annotation of the input. The most common form of this pre-processing is to add POS tags to the data. Generally these tags are used to increase coverage, by allowing the parser to treat an unknown word as a generic entry that behaves according to its POS, but they may have other uses.

In Crysmann *et al.* (2002) a POS tagger is used to prioritise lexical items in order to improve the probability of finding the correct reading earlier. In addition to POS tagging, they use two more detailed sources of lexical information to increase coverage — named entity recognition is used to create complex multiword types and semantic information contained in GermaNet is used to extend types that would otherwise be considered as generic noun types. This work is extended in Frank *et al.* (2003), where phrasal level annotation is added to the extra lexical information. The shallow processor used for the phrase level information was a German stochastic topological parser, which was argued to be a better match for the HPSG formalism than a chunker. They used the bracketing from the topological parser to order the HPSG parsing agenda, prioritising those parsing steps which would result in creating phrases that matched the topological bracketing, according to a pre-calculated mapping from topological bracket type to HPSG phrase. Confidence levels in the shallow parser were incorporated into the priority calculation, and the results showed that the best efficiency was achieved when the shallow annotation was allowed to influence up to half the original deep parser priority score.

Another experiment on German (Daum *et al.* 2003) attempted to incorporate chunk information into their Weighted Constraint Dependency Grammar and found that the chunk information improved both efficiency and coverage, but to much less an extent than POS information did. They also reported that even a small amount of inaccuracy in the chunker could halve this improvement.

Rather than merely guide the parsing order, the use of shallow tools to guide the search of the parser can be taken further to filter the use of unlikely constituents in order to increase efficiency. Grover and Lascarides (2001) describe a parser for the ANLT grammar (based on the GPSG formalism, Gazdar *et al.* 1985) that rejects readings that have assumed a different POS to the initial annotation. While they showed some sentences where this approach worked, rejecting incorrect readings, it does bring up the matter of tagger accuracy. The methods they used to counter-act issues of tagger inaccuracy involved using two taggers and allowing both tags when they disagree, and using some handcrafted patterns to correct systematic tagging errors. Despite this, they found roughly a third of the failed and incorrect parses involved tagging errors. Their results showed that using this pre-processing gave an substantial coverage increase, but this was from a very poor original coverage (2%)

to a still impractical 39.5%.

Kiefer *et al.* (1999) describe a method of filtering lexical items by specifying and checking for required prefixes and particles which is particularly effective for German with its separable verbs, but also applicable to English. Other research has looked at using dependencies to restrict the parsing process (Sagae *et al.* 2007), but the most well known filtering method is supertagging. Originally described by Bangalore and Joshi (1999) for use in LTAG parsing, it has also been used successfully for CCG (Clark 2002). Supertagging is the process of assigning probable 'supertags' to words before parsing to restrict parser ambiguity, where a supertag is a tag that includes more specific information than the typical POS tags. The next section goes into more detail about supertagging, its motivations and its effects.

2.3 Supertagging

2.3.1 Motivation

Bangalore and Joshi (1999) describe supertagging as 'almost parsing' since they find that once the correct supertags had been assigned, almost no decisions are left to their LTAG parser, speeding parsing up significantly. The motivation behind their work was based on the idea that part-of-speech taggers had long been used to reduce ambiguity in parsing and a POS tag was a category of a word that described how that word was used. Moreover POS tags could be quite accurately assigned using only local context. They reasoned that their LTAG elementary trees were also word categories that constrained word use. As a lexicalist formalism, LTAG defines complex lexical item descriptions, which were designed "to allow for all and only those elements on which the lexical item imposes constraints to be within the same description" (Bangalore and Joshi 1999). As such, many decisions that would be made by the parser were localised with the lexical item and if the correct lexical item description out of the many possible descriptions could be pre-assigned, parsing could be substantially simplified. They called these lexical item descriptions — in their case elementary trees — supertags to indicate that they embodied richer information than standard POS tags, in particular dependency information. Their initial experiments indeed showed that assigning gold supertags reduced average sentence parsing time from

120 seconds to 4 seconds over a 1300 sentence selection from the Wall Street Journal. Having proved that supertags could speed up parsing, the next step required to show the effectiveness of supertagging was to see if supertags, like POS tags, could be accurately predicted from local context.

2.3.2 Initial Experiments

The initial supertagging experiments described in Bangalore and Joshi (1994) could only manage tag accuracy of 68% using local context, which was not viable for use in parsing. Later experiments reported in Bangalore and Joshi (1999) increased the tag accuracy to 92% by using a much larger amount of training data and adding some smoothing techniques to the tagging model. Even this would not be sufficient for parsing, since any word in a sentence mistagged would render the full sentence incorrect and often unparseable. The authors suggest that allowing more than one tag could alleviate this problem, and show that accuracy rises to 97% when the top 3 tags are considered. They don't, however, report full parsing results to show whether this is effective.

Supertagging was predicted to be an effective technique for speeding up parsing based on any lexicalised formalism and Clark and Curran (2007b) show that, at least for CCG, this is the case. The supertags in their case are the 425 CCG lexical categories that appear at least 10 times in Sections 2-21 of CCGBank. The C&C supertagger uses a Maximum Entropy model, as opposed to the HMM-based model in Bangalore and Joshi (1999) and also gets around 92% single tag accuracy (when using automatically assigned POS tags as features). Rather than set a fixed number of tags to use in parsing though, they propose assigning tags with a probability within a factor, β, of the probability of the top tag. Using this method, tag accuracy is 97.86% while assigning an average of about 2 tags per word. This translates to a sentence accuracy of around 70%, compared to 37% for single tagging. Clark and Curran experiment with various strategies for integrating the supertagger into their C&C parser and achieve the best results by initially using a fairly restrictive β factor, which assigns on average 1.27 supertags per word on their development set. Only if the parse fails do they adjust the β factor to allow more tags per word for a particular sentence. On the development set, over 93% of sentences could be parsed with the most restrictive setting. Using this strategy of gradually allowing more tags as needed, they are able to parse 99.6% of section 23 of CCGBank in under 2 minutes, at 20.6 sentences per second. It is interesting to note that, when using gold supertags, the speed increases by a factor of 4, but the coverage goes down to 94.7% of the test set. The possibility of trading coverage for speed is always an issue in supertagging, since assigning an incorrect tag can make an utterance unparseable, but in this case it is not an issue of incorrect tags. For over 5% of the test set, the parser is unable to produce the gold analysis using gold tags and so it is theoretically possible to get 'better' results by assigning incorrect tags and getting part of the analysis correct. The final analysis is evaluated by measuring the precision, recall and F-score of the predicate-argument dependencies returned against those of CCGBank. As will be discussed in Chapter 3, it is difficult to compare accuracy evaluations across different formalisms, but their result of 85.45 labelled F-score appears to be state-of-the-art for CCG parsing.

2.3.3 HPSG-related Supertagging

There has also been some previous supertagging research in HPSG-based systems. Ninomiya *et al.* (2006) describe a supertagger for the lexical templates used in the ENJU grammar. With an effective tag set size of 1361, the single tag accuracy on section 22 on their converted Penn Treebank was 87.51%. In this work, the supertags were not used to restrict parser input, but instead the supertag probabilities were used in the parse ranking model. Toutanova *et al.* (2002) also use supertags as part of a disambiguation model, although in that case the supertags are taken from the set of approximately 8000 lexical entries in the ERG HPSG grammar of the time. They do not give the accuracy results of their tagger. They do demonstrate, through using gold supertags in one disambiguation model, that the correct lexical entry sequence is not always sufficient to disambiguate between parses from this grammar, giving the correct parse in only 54.59% of ambiguous sentences.

Matsuzaki *et al.* (2007) take the supertagger from Ninomiya *et al.* and integrate it into the parser in a more standard pre-processing manner, restricting the parser input to single tagged sequences and adding new sequences on parse failure. In addition, they use a CFG filtering stage between the supertagger and their ENJU HPSG parser to exclude unparseable sequences. As in the C&C parser, a high percentage of sentences (almost 95%) could be parsed using the first input tag sequence. Using supertagging and CFG filtering, the optimal settings gave an average speed of 29.6ms per sentence, or 33.8 sentences per second when parsing section 23 of their converted Penn Treebank. Differing hardware doesn't really allow direct comparison, however this is faster than the reported C&C figures, but with a lower coverage of 97.1%. On the development set they were able to get a coverage of 99.6% by changing experimental parameters, but this came with a decrease in accuracy and F-score, as well as a slower speed (slightly slower than C&C). These trade-offs are common and will be discussed in terms of evaluation in the next chapter.

Alpino is an HPSG-based grammar of Dutch which also uses a form of supertagging in its parser. Prins and van Noord (2003) describe the HMM tagger they use to assign tags from a set of 1365 lexical category classes and report standalone accuracy results of 94.69% for single tagging with this tagger, however this number can't really be compared to other results. In the first place, it is over a Dutch test set that no one else has used. Secondly, the gold standard for tags in this case was the uncorrected output of the parser. They give an example of one case where their tagger was actually correct when the 'gold standard' was incorrect, but it would be equally possible for both tagger and gold standard to be incorrect. The supertagger probabilities are integrated into the parser by throwing away potential lexical categories whose probability is not within factor τ of the top tag probability. Their optimal value of τ on the test set (216 sentences from the newspaper section of the Eindhoven corpus) produces approximately a factor of 4 speed up, while improving accuracy of lexical dependency relation recovery by about 2%. This accuracy improvement is largely due to a decrease in full sentence parsing coverage. The results of this system basically reflect those found in lexicalised English parsing systems. The main contribution of this work is that the tagger was trained on the uncorrected output of the parser, circumventing the data scarcity issue often a problem in combining statistics with deep linguistic processing.

Specifically related to the grammars and parser that will be used in this thesis, Blunsom (2007) develops a supertagger for assigning lexical types from the ERG and Jacy grammars (615 and 360 tags respectively). This tagger, which slightly outperforms the C&C tagger when tagging CCG categories from CCGBank, achieves single tag accuracy of 93.7%

for Jacy and 90.3% for the ERG. No full parsing experiments were carried out with this tagger, but examination of the test data showed that many complete, but non-preferred analyses could be removed by limiting the lexical types to those produced by the tagger. A limited scope experiment that manually restricted the lexicon according to the predictions of the parser showed that properly integrated supertagging should also be effective for this setup.

2.3.4 The Tags

While all the above work can be categorised as supertagging, the actual supertag set differs by formalism and even by researcher. Comparing only English tag sets, the tag set size varies from around 300 LTAG elementary trees, to 8000 HPSG lexical entries, as summarised in Table 2.1. The Penn Treebank POS tag set, by contrast, has 48 tags.

It's not only the tag set size that differs, but also the distinctions made in each tag set. In LTAG, for example, the word *price* has different elementary trees in *This is the price.*, *What is the price?*, *The price includes breakfast.* and *The price war rages.* CCG, on the other hand, uses the same category for the first three sentences, but has a separate category for *price* used as a bare noun phrase (without determiner). For both CCG and LTAG, different inflections of the same verb (i.e, *includes* and *included*) have different sets of possible supertags. Examples of supertags of each type are shown in Figure 2.4.

For HPSG, the situation is more complex since different parts of the lexical information are encoded in different places. As mentioned in Section 2.1.1, the building blocks of an HPSG analysis are the *word* signs, but words come from the combination of lexical types, which encode subcategorisation information; lexical rules, which add information triggered by inflection; and lemma specific information such as phonology and lexical semantics. Previous work in HPSG supertags has used different combinations of this information: Blunsom (2007) used the lexical types, Ninomiya *et al.* (2006) used lexical templates, which combine lexical types and lexical rules, and Toutanova *et al.* (2002) used lexical entries, which combine lexical types and lemma specific information, but ignore inflection. This variation has, in part, been due to characteristics of the parser and grammar implementations. In the DELPH-IN grammars, used by Blunsom and Toutanova *et al.*, lexicon entries are lemmatised and lexical

	Formalism	# Tags	Tag shape
Bangalore and Joshi (1999)	LTAG	300	elementary trees
Clark and Curran (2007b)	CCG	425	lexical categories
Blunsom (2007)	HPSG	615	lexical types
Ninomiya *et al.* (2006)	HPSG	1361	lexical templates
Toutanova *et al.* (2002)	HPSG	8000	lexical entries

Table 2.1: Supertags across systems

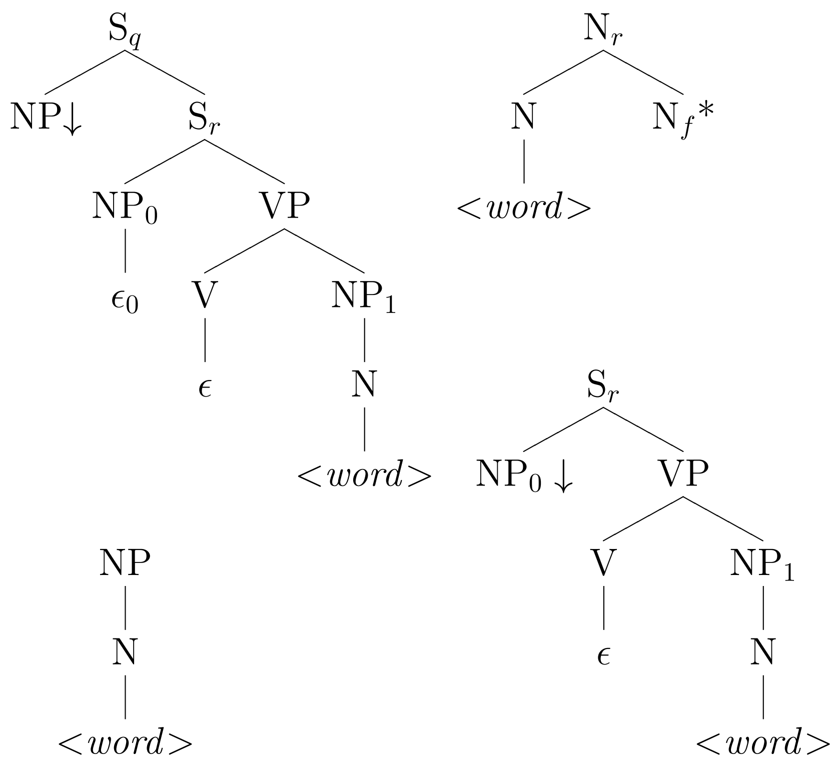
rules are applied on-the-fly before lexicon lookup. In the ENJU grammar, used by Ninomiya *et al.*, the lexicon entries are words indexed by the combination of the inflected word and the POS tag. Chapter 4 will further explore these differences and the effects they have.

Other factors of the parser and grammar affect the way that different information can be integrated into the parsing process. The next section describes some of the concrete problems that had to be overcome in order to use any tagger with the PET parser.

2.4 Integration Issues

All of the experiments described in this thesis involve combining information from disparate sources into the PET parser. Before any integration experiments could be carried out however, there was one issue, touched on in Section 2.2, that needed to be overcome. Integration requires some common element on which to match, and the obvious element in this case is the word. The problem is that the definition of *word* is not straightforward.

Most English treebank-derived parsers have standardised by default on the tokenisation used in the Penn Treebank, since that is the data they learnt from. In linguistically-motivated grammars like the ERG or GG (for German), the decision regarding the definition of a word was made according to the linguistic analysis given by the grammar writers. In the case of the ERG, this analysis has given rise to a tokenisation that differs from that of the Penn Treebank in certain ways. The most obvious is that punctuation is not treated as a separate token except for particular punctuation marks (such as colons and certain dashes) that are analysed as contributing an independent elementary predicate to the semantic analysis. In general, punctuation is treated as an affix to a word, and is processed using lexical rules, similar to those for inflection.



(a) LTAG elementary trees

N
N/N
NP\NP

(b) CCG categories

Figure 2.4: Supertags for the noun *price*

Other tokenisation differences include splitting hyphenated compounds, but not splitting contracted negations such as *don't*.

All of the DELPH-IN grammars include preprocessing rules that describe how to tokenise their input (except Jacy, which includes directions for using the Japanese morphological analyser ChaSen (Matsumoto *et al.* 1999) as a preprocessor). However, these rules assume a raw text input and cannot be applied to pre-tokenised input. In order to use the output of a tagger and still input the tokenisation that the parser expects, there are a couple of solutions. The obvious one is to retrain the tagger using the parser's tokenisation so that tagging and parsing operate over the same tokens. However that requires a large annotated training set with appropriate tokenisation. It might be possible to convert a training set from one tokenisation to another, but that would be an inexact process adding noise to the data. Furthermore, one of the aims of hybrid processing is to integrate information from different sources, even when the original raw information is not available. Having to pre-process all the raw information negates some of the benefits of integrating off-the-shelf tools.

Assuming then that the goal is to map information between different token sets, this can be done in a pipeline, or by merging. In the pipeline scenario, the input is first tokenised according to the rules of the tagger, then tagged, and then the differences in the tokenisation rule sets are extracted and this difference set is applied to the tagged token. If the differences really only required re-attaching the punctuation to the neighbouring token, this would be simple, but the differences are in reality more subtle than that. It is possible to write a highly accurate set of rules to map between tokens, but it requires a different set of rules per grammar, and possibly per grammar version since the grammar writers may change tokenisation assumptions as their linguistic analysis evolves.

The merging process, on the other hand, leaves control of tokenisation assumptions with the grammar writers by making use of the preprocessing rules released with the grammar. In this scenario, the input is tokenised twice, once according to the tagging rules, and once using the parser's rules. Then, using the parser tokens as the controlling set, the tagger's output is consulted and added to the appropriate parser token. This process is made much simpler by the agreement among DELPH-IN members on the necessity of *characterisation information* in analyses. This refers to recording the character span of each token, *with respect to the original string*. The idea that details of the analysis can always be related back to the appropriate part of the original input is useful in all forms of integration, including post-parsing analysis merging and integration with external lexical resources, and makes merging between tokens mostly straightforward. Characterisation information is also added to the tokens for the tagger as the input is tokenised and hence the tagged token(s) that correspond with the parser token can be identified by their character span and all the tags added to the parser token. Certain heuristics operate to filter the tags from multiple tokens, throwing out punctuation tags (which would be ignored by the parser anyway) and not keeping duplicate tags. This is still a fuzzy solution, since no correction is made to tag probabilities when assigning tags from two different tokens to one, but short of annotating a new training set to acquire the correct model, it was thought the best solution for an unexpectedly complicated problem. A very recent version of the ERG has implemented a different solution within the grammar, where the grammar writers have developed preprocessing rules that assume a Penn Treebank tokenisation and apply the necessary changes to produce the tokenisation the grammar requires (Adolphs *et al.* 2008). The tags, however, are still handled in a similar way to that described here, with heuristics determining which tags to keep when tokens are split or merged, and with punctuation tags being discarded.

2.5 Summary

The PET parser is an efficient unification-based parser which, when used with the hand-crafted DELPH-IN HPSG grammars, can produce deep detailed syntactic and semantic analyses of input in several languages. A number of aspects of this parser and grammar combination that contribute to the rich analysis, at the same time bring disadvantages when compared with other parsing systems whose grammars are derived from treebanks and may produce shallower analyses. Hybrid processing attempts to overcome disadvantages of one system by combining it with others with different advantages and disadvantages. In particular, one form of hybrid system involves integrating different sources of information into a parser to increase robustness. Supertagging is one instance of this integration that is particularly appropriate to lexicalised grammar formalisms. Previous work has shown that supertagging is effective in improving parser performance for LTAG and CCG parsers, and some preliminary work suggests that it will also be effective for hand-crafted HPSG grammars such as those used by the PET parser. Before any work on integrating information into PET can be undertaken, it is first necessary to be able to relate the information between different sources. The final section of this chapter outlined the problem of disparate tokenisation assumptions and the method used to overcome this issue.

In order to ascertain whether the proposed integration of statistical information into the PET parsing process is effective, it is necessary to evaluate the performance of the parser with and without the information. Parser performance can be measured against multiple criteria and in a variety of different ways. The next chapter examines how parsers have been evaluated in the past and defines the evaluation criteria that the PET parsing system will be measured against.

3 Parser Evaluation

One of the goals of this thesis is to measure how lexical statistics can help the parsing process, but this goal needs further definition. What exactly does it mean to 'help' parsing, and how do we measure this help? This chapter examines parser evaluation: different evaluation criteria; accuracy evaluation metrics based on different representation; inter- as well as intra-parser evaluation; and the relationship between evaluation metrics and the goals of parsing and evaluation. The second half of the chapter, using the ideas from the first, considers the evaluation metrics used with the PET parser and outlines the evaluation metrics that will be used for the rest of this thesis, including a new granular accuracy evaluation method based on the parser MRS output format. Baseline experiments without any input annotation are described, providing results with which subsequent experiments will be compared.

3.1 Background

Parser evaluation is, in terms of Gaizauskas (1998), evaluation from the perspective of researchers rather than users or funders, and on a task that is primarily user-transparent, rather than user-visible. Evaluation of this kind is necessary to direct and focus language technology research. For example, without breaking down a question answering system into its component parts, it is next to impossible to discover what causes an incorrect answer to be returned, and so there is no way to know where improvement efforts should be focussed. On the other hand, the disconnect of a parse with the desired result makes defining a useful evaluation scheme much more difficult, since what makes a 'good' parse depends on how that parse will be used towards producing the end result. The ill-defined goal of parsing has produced many parse evaluation schemes.

Evaluation itself can have many goals, even focussing just on evaluation by and for researchers. The goals listed below, while overlapping significantly, all give rise to slightly different evaluation needs.

- Task suitability: evaluating a parser for the benefits it brings to a wider task, which may have constraints on runtime, memory usage or accuracy required.

- Parser comparison: comparing among parsers (potentially with different mechanisms and different outputs) to select 'the best'.
- Parser development: evaluation to focus development efforts on flaws in the grammar or parser, while not making changes that decrease parser performance.
- Parameter tuning: selecting the optimal configuration of a particular parser.

Ideally the first goal of task suitability should inform any parser evaluation, but that is not always straightforward, since the requirements of the (possibly hypothetical) task are not always known. Some effort however should be made to define the goals both of parsing and of evaluation.

One obvious evaluation criterion is the accuracy of the analyses, but there are other factors to consider. Section 2.1 mentioned that robustness is the main limiting factor proscribing the use of the PET parser in applications. As discussed there, there are multiple reasons for parse failure. Resource exhaustion is one, and a parser's resistance to this cause of parse failure can best be measured by evaluating parser efficiency. In computer science, efficiency is considered in terms of both time and space, or more specifically runtime and memory usage. At a fine-grained level, these aspects are difficult to compare between different runs because they can both be affected by differences in hardware, arbitrary implementation decisions and run conditions (such as what sort of output is requested). In a controlled scenario, however, they can give a picture of differences between different configurations of a parser and, at a coarser grained level, they are an important part of a parser's suitability for a task. Any comparisons between parsers should define exactly what is being measured. For example, is it time per sentence successfully parsed or per sentence input? Another variation seems to be measuring complete runtime including loading of models versus parsing time only.

The other causes of parse failure mentioned were ungrammatical and extragrammatical input, however, without prior grammaticality judgements on text input, these are hard to tease apart. In fact, most recent parsing research operates on the assumption that they shouldn't be, that all input should be given an analysis. Under this assumption, a common evaluation criterion is coverage — the percentage of input utterances that receive a parse. This directly measures robustness, including failures resulting from resource exhaustion when resources are limited. When grammaticality judgements are available, and the parser is supposed to reflect these grammaticality judgements, it makes sense to also include the criterion of overgeneration — the percentage of inputs that receive an analysis when they shouldn't. In this case, the definition of coverage is refined to the percentage of grammatical sentences that received a parse.

Carroll *et al.* (1998) give an overview of the state of parser evaluation up until eleven years ago. They differentiate between metrics that require an annotated corpus and those that don't. Characteristics of a parser like coverage and efficiency can be measured without the need for an annotated corpus. Both of these measures are useful in parser or grammar development, but do not give a full picture of how good a parser really is. For that, some sort of gold standard annotated corpus is required to measure parser accuracy. An obvious measure of accuracy is how many sentences received the correct analysis according to the gold standard, known as *exact match.* This metric is occasionally reported (e.g., Briscoe and Carroll 1993, Yamada and Matsumoto 2003) and Magerman (1994) claims it is the only reliable, concrete evaluation. Against that however is the problem that exact match cannot distinguish between a truly awful parse and one that is very close. The more detailed a parse is, the more likely that one small detail is incorrect. In order to still be able to use detailed analyses, we need to know how correct they are, and this can be done by making more granular accuracy judgements. The standard method for doing this defines sub-sentential elements about which a binary correctness judgement can be made. Granular accuracies are usually reported as the precision, recall and f-score of these elements, with the following definitions:

precision

$$\frac{\text{number of elements correct}}{\text{number of elements found}}$$

recall

$$\frac{\text{number of elements correct}}{\text{number of elements in gold standard}}$$

f-score the harmonic mean of precision and recall

$$\frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

The following sections look in more detail at the history of granular accuracy evaluation.

3.1.1 Phrase Boundaries

PARSEVAL (Black *et al.* 1991) was the first serious attempt to define a representation that would allow different (English) parsers to be compared with each other. The definition they came up with utilises a lowest common denominator approach, describing a set of steps that should be applied to a phrase structure tree to remove all those phenomena for which the correct analysis was debatable. Hence, any instances of "not", pre-infinitival "to", auxiliaries, null categories, possessive endings and word-external punctuation are removed, and then parenthesis pairs that now contain less than two words are deleted. Finally, scores are calculated based on parenthesis only, ignoring constituent labels. Two things can be taken from that work. The first, clearly, is the evaluation metric that has been used ever since, despite its obvious shortcomings, as a standard parse evaluation metric. The second thing is the reason that such a metric was defined: comparing parses according to constituent bracketing and labelling is very difficult. Despite a general agreement in the meaning of a sentence, each grammar makes different decisions and distinctions in how they represent details of the syntax. A paper by Gaizauskas *et al.* (1998), seven years later describes an even more impoverished annotation scheme to be used purely for parser evaluation. It is clear in this work however that the scheme is intended only as a minimum standard for parsers and hence they dispense with precision completely for evaluation, reporting only where a parser failed to return the minimal markup, or where it violated the minimal constraints thus annotated.

Subsequent work in parsing often refers to the PARSEVAL metric, but in actual fact uses a less lenient version than that originally published. The most common metrics are *labelled* precision, recall and f-score of brackets, though unlabelled bracketing scores and bracket crossing scores are also often reported. The EvalB program (Sekine and Collins 1997) is used by many people in order to report comparable results. One reason that the labelled constituents can be more easily used now than when PARSEVAL was defined is that much of the recent parsing research has focussed on treebank-derived parsing, using the Penn Treebank (PTB: Marcus *et al.* 1993) for training and testing. Hence the annotation on the Penn Treebank has become the default standard for parsing in English. While this has led to encouraging progress in statistical parsing, it is not without its problems. In the first place, there are disagreements about the quality of the PTB annotation which contains what are now considered idiosyncratic analyses of, for example, noun phrases. There are suggestions that all the progress in statistical parsing has led to a state where we can now reproduce the shortcomings in the PTB with high accuracy. Furthermore, even if the linguistic annotations of the PTB were correct, questions remain as to what use they are in NLP systems.

A workshop entitled *Beyond PARSEVAL* was convened in 2002. This workshop recognised that PARSEVAL was biased towards shallow treebank grammars for English and one of the stated aims of the workshop was to discuss a "*new, uniform evaluation metric which provides a basis for comparison between different parsing systems, syntactic frameworks and stochastic models, and how well they extend to languages of different types*" (Carroll *et al.* 2002). Various comparisons and alternatives were presented. The leaf-ancestor metric (Sampson and Babarczy 2002) calculated the tag or label path to each word, and then averaged the edit distance of each tag path over all words. Roark (2002) presented a similar algorithm, but generalised to look at the edit distance between different representations of the parse including derivation rule sequence. In both cases, the new metrics attempted to mitigate the problem of counting an incorrect attachment multiple times, a common criticism against PARSEVAL. Interestingly, they also both noted that these new metrics made it easier to discover systematic errors. Given that one object of parser evaluation is for the benefit of the parser developer, this can be a useful feature. Roark's main point however was a comparison between the metrics of precision and recall versus edit distance, rather than advocating for a particular parse representation. As he pointed out, edit distance could also be used to measure accuracy in terms of constituent spans in the PARSEVAL model, as well as in terms of dependencies. Other researchers at this workshop presented comparisons between the PARSEVAL metric and those based on dependency-type relations, as described in the following section.

3.1.2 Dependencies

The usefulness of phrase structure annotation was questioned directly in Lin (1998): "*. . . the purpose of parsing is usually to facilitate semantic interpretation. No semantic interpretation algorithm makes direct use* *of phrase boundaries.*" Lin went on to propose syntactic dependencies as a better granularity to evaluate against. He argues that dependency evaluations more closely coincide with the intuitions of human evaluators. While the algorithm presented matches unlabelled dependencies, comparable to the unlabelled bracketing of PARSEVAL, Lin suggests that, given that an equivalence mapping between relations can be created, matching labelled dependencies is possible. Indeed, one benefit he claims dependency evaluation has is the ability to selectively evaluate the relations of interest, for example determining how well a parser predicts prepositional phrase attachment or the subject–verb relation.

Carroll *et al.* (1998) outline a new evaluation method based on grammatical relations (GRs). These relations are very similar to the relations encoded by the dependencies Lin describes. According to the authors, the differences are: a single dependent may have two heads, thereby making a graph rather than a tree; arguments 'displaced' by movement are associated with their underlying relation; semantic arguments realised syntactically as modifiers can be represented; and arguments that are not lexically realised can be included (e.g. subjects in pro-drop languages). These points do differentiate them from Lin's work, but not from all dependency representations. Traditional syntactic dependencies are surface based, but there are annotations which are called dependency-based but are even further from surface form than the GRs, such as those used in the Prague Dependency Treebank (Böhmová *et al.* 2003). The effective differences of the GR based proposal from other dependency representations are the use of relation subtypes and the definition of a relation hierarchy to allow lenient evaluations.

While both Lin (1998) and Carroll *et al.* (1998) promote dependency-based evaluation because of its advantages in cross-parser comparison, Collins (1999) uses word–word dependency evaluation for stand-alone evaluation of his treebank parser, trained on the Penn Treebank. His motivation for doing so is that this allows a finer-grained evaluation, capturing attachment accuracy. This reinforces the idea that dependencies are better able to show what a parser should be attempting to return. Collins uses a set of heuristics to determine the head word of each constituent in a Penn Treebank tree and calculates dependencies based on these. Since the same mapping is carried out for the gold standard and the parser output (and indeed is used within the parser), there is no problem with mis-matching, although no evaluation is given of how the heads so chosen would relate to those a linguist would select. However, at least for English, the position of head word of a phrase is generally consistent, given phrase type, and so these heuristics may be supposed to give accurate results.

Multiple groups at the Beyond PARSEVAL workshop looked at evaluation metrics measured over dependency or grammatical relation type representations. Clark and Hockenmaier (2002) compared evaluation of their CCG parser using the PARSEVAL metrics, the dependencies of Collins (1999) and the dependencies that a different CCG parser (C&C: Clark *et al.* 2002) produces natively. They find that the flat nature of the Penn Treebank, compared to the binary branching of CCG trees, leads to an automatic decrease in bracket precision for CCG and also makes PARSEVAL too lenient towards mis-attachments produced by Penn Treebank parsers. The Collins dependency measure, in contrast, is neutral to the branching factor of trees, since the number of dependencies is related only to the number of words and they find that the unlabelled dependencies allow a valid comparison with Collins' work. There is still a problem comparing labelled dependencies however, since the node labels used by CCG are very different from those used by Collins and include information about the sentence structure. This leads to one error being propagated to every word in that constituent, a similar problem to that caused by using bracketing and one that dependency-based evaluation was meant to circumvent. The C&C CCG dependencies are 'deep' dependencies, encoding argument positions even when arguments are displaced due to raising, extraction etc. and hence are quite similar to those of Carroll *et al.* (1998). However, as with the other CCG dependency evaluation, the relations here are CCG constituent labels which are very detailed and encode how a derivation was calculated, rather than just represent the final result. As such, while it would be a useful tool for focussing further development work on a CCG parser towards problematic structures, this dependency style would need to be generalised to compare with work in anything but the CCG formalism.

Another evaluation metric comparison at Beyond PARSEVAL was by a group (Crouch *et al.* 2002) that worked with a different non-Penn Treebank parser, the LFG-based XLE parser (Maxwell and Kaplan 1993). They compared between evaluation at the level of a feature specification of their native f-structure (a preds-only version that strips out some grammatical detail) and the grammatical relations suggested by Carroll *et al.* (1998). Their second evaluation involved mapping from the f-structure to grammatical relations. They report that both metrics appear to display similar behaviour with respect to upper bound, lower bound and error reduction, but that the grammatical relation metric is systematically lower than the other and they attribute this to the mapping process. Mapping from one representation to another is a large part of any comparative evaluation and even between representations that appear to encode the same information, small differences in tokenisation, lemmatisation and granularity, as well as actual differences of analysis opinion can lead to significant differences in scores. As mapping is a recurring theme within parser evaluation work, the next section is devoted to summarising some of the methods, problems and results related to inter-representation mapping.

3.1.3 Mapping Between Representations

PARSEVAL was designed as a lowest common denominator standard, ignoring any distinctions for which there was no consensus. Despite this minimal standard, mapping rather than deleting was still required in some circumstances. One early attempt to use the PARSEVAL metric (Grishman *et al.* 1992) describes the rewriting they had to do, even given the specified deletions, to compare their Linguistic String Theory-based parsing (subject, verb, object) to the X-bar theory based treebank.

In proposing dependencies as a medium for evaluation, Lin (1998) also defines an algorithm for mapping from a phrase structure tree to a dependency representation. This algorithm uses heuristics to select the head word on any constituent (similar to Collins (1999)) and is quite English specific. No full scale evaluation of the fairness of this mapping is given. More generally useful, Lin (1998) also provides a method of mapping between parses that use different tokenisation. As tokenisation is a major problem in parser comparison, this step is frequently necessary. Lin's method, for each span where tokenisation differs, takes the largest token as the standard and ignores dependencies within that. Since this is open to abuse (by calling a whole constituent one token), token boundaries are enforced at white space. This is a reasonable heuristic but fails when multi word expressions are given special treatment by a parser (as is the case with many lexicalised parsers).

Crouch *et al.* (2002) detail some of the problems they encountered when attempting to map between two representations that might be supposed to encode almost the same information, LFG f-structure features and the grammatical relations of Carroll *et al.* (1999). As they calculated the mapping between gold standard annotations of the same data set, they expected a straightforward and deterministic mapping. Apart from tokenisation problems, 'standardisation' of the lexical items caused issues. The grammatical relation (GR) gold standard used the uninflected form of a word as the head (e.g. *walk* for *walking*) which could usually be recovered from the LFG f-structure, but in some cases there were inconsistencies where, for example, *should* was mapped to *shall* in the GR standard but not LFG, and *himself* to *he* in LFG but not GR. Added to this, in the GR standard any American spelling was normalised to the appropriate British representation. At a higher level, there were problems because the authors were mapping from the f-structure only, since that was supposed to encode the dependency relations, however they found the GR standard actually encoded a mixture between surface phrase structure and underlying dependencies. Hence the f-structure, by abstracting away surface differences, lost some of the information necessary to distinguish clausal and non-clausal subjects, as well as the surface representation of tense, aspect and mood, which are instead reported as semantic features in LFG.

Learning from the problems they found in the above mapping task, a new dependency bank, the PARC 700 Dependency Bank (DEPBANK: King *et al.* 2003) was created with the aim of abstracting away from those surface structure features recorded by Carroll *et al.*. Kaplan *et al.* (2004) then used this dependency bank to evaluate their own parser and the Collins statistical parser (Collins 1999), requiring a mapping from the shallower Collins parser output to DEPBANK format. The mapping they perform replaces surface form words with their uninflected version and makes some systematic changes for head selection in coordinations and sentential complements. They also compensate for the difference in treatment of auxiliaries. The Collins parser distinguishes objects, adjuncts and subjects, but not exactly as LFG does. No attempt was made to adjust the subjects of coordination or of matrix verbs, since this was considered to be adding extra information that the parser did not know. While the authors released the results of the mapping, it is still hard to evaluate the fairness of the subsequent comparison. As reported, the Collins parser had an f-score 5% lower than the best result of the XLE parser, but it is not clear how much of that gap is due to the mapping process.

Another mapping to DEPBANK was described in Burke *et al.* (2004).

In this instance, the goal was to evaluate an automatic f-structure annotator that produced f-structures from Penn Treebank trees. Despite the fact that DEPBANK and the annotator are linked to the same linguistic formalism, LFG, the conversion was not straightforward. The first stage of mapping added the DEPBANK named entity markup to the parse trees to be annotated. Then, after annotation, post-processing was carried out to rename features, in some cases collapsing distinctions made in DEPBANK (e.g. allowing ADJUNCT to map to either ADJUNCT or MOD). The addition of extra features was also required, using heuristics to determine properties such as number and statement type from the base annotation and the lexical item. Finally XCOMP Flattening was used to transform the automatic annotation of auxiliaries to the PRED and TENSE representation used in DEPBANK. After all this, the f-score was still significantly lower than that achieved over their own evaluation set, DCU 105,[1] a set of hand-constructed LFG analyses for 105 randomly selected sentences from Section 23 of the Penn Treebank. While overfitting on the DCU105 could contribute to this (and was one motivation for this new evaluation attempt), as with the Collins parser experiment, it is hard to quantify how much of the error can be attributed to the mapping process. Examination of the results showed one issue to be treatment of hyphenated words, which are commonly, but not always separated in DEPBANK, which was not considered in the mapping since there is no way to decide when to split deterministically. This was not a parsing task, so the results can not be compared to the early work, but they still underline some issues in mapping: both the difficulty (even with the same formalism) and the problem of distinguishing errors due to conversion and errors due to the task being evaluated.

Briscoe and Carroll (2006) also attempt to evaluate against the PARC 700 DEPBANK, evaluating a parser that produces grammatical relations (GRs). They find (mirroring Crouch *et al.* 2002), that DEPBANK annotation is too close to its LFG roots and re-annotate with their own system. In some cases this involves simplifying or omitting the DEPBANK annotation, though there are instances of annotating according to different linguistic philosophies. This is an alternative to mapping (particularly as they have released their annotations alongside the original DEPBANK annotations) but it doesn't allow numeric comparison, merely points out the disagreements over what should be evaluated.

[1]Available at `http://www.computing.dcu.ie/research/nclt/gold105.txt`.

Clark and Curran (2007a) use the Briscoe and Carroll re-annotation (DepBank: Briscoe and Carroll 2006) to evaluate their CCG parser. Again they find that an apparently straightforward mapping task is a time-consuming and non-trivial exercise. In many cases, as in the LFG to GR conversion, mapping required converting from a deep semantic relation to one more surface related. As an attempt to explicitly measure what effect mapping errors have on the final results, they run the conversion script they devised on *gold standard* CCG annotations for the same data set as that of DepBank. Comparing these dependencies to the DepBank annotations, they achieve 84.76% f-score, after a lot of effort had gone into the conversion process. Since this is what they achieve on perfect parses, this provides an upper bound of what the parser could hope to achieve over DepBank. The results from their actual parser give an f-score of 81.86%, less than 3% below the upper bound. They do not report what results they achieve over this data set against the CCGbank dependency annotations, but over the entire Section 23 (from which the DepBank sentences come), the parser achieves an f-score of 85.45%. It is not clear what these results mean, but their claim that any evaluation that involves mapping should report an upper bound on the conversion has merit.

Miyao *et al.* (2007) take on the challenge given by Clark and Curran to evaluate on DepBank, and provide a conversion upper bound along with the results. They convert the output of their ENJU HPSG-based parser, along with three Penn Treebank style parsers: Charniak and Johnson's reranking parser, Charniak's parser and the Stanford parser. Results could only be called inconclusive. While ENJU achieves the best f-score compared to the three treebank parsers and also Clark and Curran's CCG parser, they also have the highest upper bound from the conversion process. Table 3.1 summarises the relevant results they give. Is a parser that comes closer to its upper bound a better parser than one with a higher score?

3.1.4 The Score

Any sort of quantitative evaluation ends up producing one or several final numbers. The different methods of calculating these final numbers can produce significant differences in the final number, but this fact is often disregarded in reporting results. One aspect of this is clear definitions of

Parser	Upper bound on f-score after conversion	Parser f-score after conversion	Absolute difference from the upper bound
Enju	87.14	82.64	4.50
C&J parser	74.54	72.81	1.73
Charniak parser	74.54	72.65	1.89
Stanford parser	74.54	69.44	5.10
C&C parser	84.76	81.86	2.90

Table 3.1: Microaveraged scores over the DepBank GRs from different parsers, after mapping. From Miyao *et al.* (2007)

the evaluation criteria: accuracy, precision, recall and coverage should all be clearly defined, particularly when examining parsers that do not always produce an analysis. Recall of sub-sentence elements, for instance, varies significantly depending on whether the number of gold elements includes those from the inputs for which no analysis was returned. Rimell and Clark (2008b) talk about a more subtle factor of how the selection of which granular elements to count can affect final scores as well as partial credit. Their work is based on the assumption that the real linguistic objects we wish to measure are syntactic phenomena, and so they describe the effect, in three different granular relation-based evaluation schemes, of mis-identifying all or part of various syntactic phenomena, such as passive and coordination constructions. One highlight that comes from this work is that relations are not independent and hence getting one aspect of an analysis correct can make it (almost) certain that another aspect of the analysis is also correct. This leads to the criticism that very fine-grained evaluation schemes have the effect of double counting certain correct (or incorrect) decisions that a parser might make. The flip side of this is that coarser grained schemes do not allow for the full range of partial credit to be counted. One solution to this issue would be to count only those relations that are at least partially independent, however this actually depends on the parser formalism and implementation. For cross-framework comparison it is impossible. The other 'solution' is to reframe the problem. Going back to the earlier ideas of this chapter, evaluation should be about evaluating how close the parser output gets us to the parsing goal. If, indeed, the goal of parsing is to identify completely certain syntactic phenomena, then that is what should be counted. If certain phenomena, or aspects of phenomena are more important to get right, then they should have a heavier weight in the calculation of the final score. The re-occurring problem is that evaluation has generally been framed as how well does the parser do what it does, rather than how well does the parser do what needs to be done. While both perspectives have some validity for internal parser assessment, when comparing parsers or evaluating suitability for a task, the second, more independent perspective should be the guiding question.

3.1.5 Application-based Evaluation

One inter-parser evaluation method that directly addresses the question *how well does the parser do what needs to be done?* is to use application performance as a metric. By substituting different parsers into place in an application, the performance of the application provides a direct means to say which parser performs better in that application. Miyao *et al.* (2008) take eight different parsers, and perform such a task-oriented evaluation, where the task is to identify protein-protein interactions in biomedical texts. They use parsers with different default output formats, including phrase structure trees, dependencies and predicate argument structures and convert each (where possible) to five different representations: Penn Treebank style phrase structure trees, three different dependency formats and the predicate argument structure used by the ENJU parser. Features were extracted from each parser and representation combination, and fed separately to SVM-based machine learning classifiers. Their results showed that, for the same training data, all parsers had approximately the same accuracy, and all representations were about the same, except the phrase structure trees that led to lower accuracy. There was significant differences in the speed between parsers, with dependency parsers being much faster than all the others. This is an interesting result, but the methods raise certain questions about their conclusion that all the parsers were equal. Both the conversion to different representations and the feature extraction mechanisms affect the final performance, as the authors acknowledged when they suggested that better results might be achievable for the phrase structure tree representation if different features were extracted. By limiting the features to something that all parsers can produce, this evaluation method can artificially limit the contribution of a parser. Defining a task-based evaluation that this criticism could not be levelled against is very difficult however, and this work provided meaningful numbers to application developers showing how parsing could help their application.

3.1.6 The Lessons Learnt

A recurring problem from the above discussion is that everyone wants to evaluate something different, usually something that could be summarised as "what my parser can produce". Hence people working with Penn Treebank parsers want to evaluate how well their parser can recreate the Penn Treebank, dependency parsers want to evaluate syntactic word-to-word dependencies, and deep parsers want to evaluate deep or semantic dependencies. This comes about because of the issue raised earlier in this chapter: the disconnect between the parser and how the parser output will be used.

While there are no silver bullet solutions to this problem, many of these issues can be mitigated for a particular situation by carefully defining the goals of evaluation and the (hypothesised) goals of parsing. The second half of this chapter describes the evaluation metrics that will be used for the remainder of this thesis. In this work, the goal is to compare variations of the same parser to judge the effect of specific modifications and as such, there is no need for mapping to a generic representation. However, the envisaged goal of parsing is to use the analyses in an application, and so the evaluation metrics need to be easily understood by an application developer. While the exact application is not specified, we assume an application that requires some semantic information rather than merely a statistical language model. By evaluating robustness, efficiency and accuracy separately, we hope to allow application developers to make an informed decision depending on the particular requirements of their application.

3.2 Evaluating the PET Parser Output

3.2.1 The Treebanks

As Carroll *et al.* (1998) say, full parser evaluation requires an annotated gold standard and both the English and Japanese DELPH-IN grammars that are used with the PET parser come with a set of treebanks that can be used for this purpose. As explained in Chapter 2, an HPSG analysis is much too complex to be annotated by hand, and so these treebanks are produced by parsing the input items using the PET parser, and then having an annotator select the correct analysis using the LinGO Redwoods treebanking environment (Oepen *et al.* 2004b). Unlike the Penn Treebank (or others such as the Prague Dependency Treebank or the German TiGer Corpus), the treebanks created in this fashion are *dynamic* treebanks. This means that as the underlying linguistic theory evolves and matures, the annotations can be easily updated to reflect any changes. In order to facilitate this, the annotations are recorded as a set of decisions based on so-called parse discriminants (Carter 1997).

The annotator is initially shown all analyses produced by the parser along with a list of discriminants that distinguish between sets of parses. By making binary decisions on the correctness of these discriminants, the set of analyses is whittled down until only one parse remains, at which point this parse can be accepted or rejected. When the grammar is changed as the grammar writers refine their analysis, the corpus can be reparsed and all the old annotation decisions still applicable can be automatically applied to the new corpus. Any unchanged analysis is kept and the annotator only needs to see parses that are affected by the change. Additionally, many of the old decisions will still be relevant even for those parses and only a few new decisions should be required of the annotator.

There are many treebanks released with the English Resource Grammar (ERG: Flickinger 2002). Some of them are test suites specifically written to cover a set of linguistic phenomena and the items in these data sets tend to be short in order to better focus on the phenomenon of interest. The other data sets are taken from real world corpora and reflect real world language usage. It is from this second group that we select the sets that will be used for evaluation throughout this thesis.

An important artifact of the way these treebanks are produced is the fact that not all test items have a gold analysis. Only those items for which, firstly, the parser produced an analysis and, secondly, the annotator decided that at least one of the analyses was completely correct will have a gold analysis recorded. It would, of course, be possible to only release those items that have a gold analysis, but these data sets are used, among other things, to measure coverage of the grammar which requires a fixed test set against which improvements can be measured. This leads to three categories of test items within a test set: those with a gold analysis; those parseable, but with no completely correct analysis; and the unparseable items. Table 3.2 describes the test sets we will be using, including the number of items, their average word length, the number of items with a gold analysis, the number of items with any analysis (*Parsed*) and the ambiguity of these parsed items. This ambiguity figure is the average number of analyses per item and can be considered both a measure of grammar precision and an indication of the difficulty of parse ranking. Lower ambiguity shows a 'tighter' grammar, one that discards ungrammatical analyses, leaving it to the parse ranking model to discard possible but unlikely analyses. Since language is genuinely ambiguous, this number will almost always be greater than one, even for the perfect grammar, and will in general be higher for longer test items. In the creation of this gold standard data the number of analyses is limited to the best 500 according to the parse ranking model released with the grammar. As such, the ambiguity numbers reported here are not true ambiguity, but still give a representative picture, since many items will have fewer than 500 analyses.

Of the four test sets shown, the first two consist of Norwegian tourist data from the LOGON English-Norwegian machine translation project (Oepen *et al.* 2004a). The *jh5* set is part of the training data for the parse ranking model, while *jhpstg*$_t$ is material from the same source, but not used in training the statistical model. This data has been used to test the grammar for many years now and so the lexical coverage of the grammar is high for these sets. The last two sets are web data — in both cases edited text and hence they don't have the same high proportion of ungrammaticality that could be expected in less formal web data, but they both contain a lot of technical vocabulary and other uncommon words.

Test Set	Items	Word Length	Gold	Parsed	Ambiguity	Comment
jh5	464	12.92	413	442	196.73	Tourism data
jhpstg$_t$	814	13.45	739	784	202.62	Tourism data
ws02	946	17.35	717	853	269.56	Wikipedia articles
cb	769	21.63	503	649	317.09	Technical essay[a]

[a]The Cathedral and the Bazaar, by Eric Raymond.
Available from `http://catb.org/esr/writings/cathedral-bazaar/`

Table 3.2: Summary of English test data sets, showing the number of items in each set, the average word length, the number of items for which a gold analysis is available and the number of items for which at least one parse was possible. Ambiguity is the average number of parses found for each item, where the parser was limited to a maximum of 500 parses.

3.2.2 Coverage and Efficiency

In order to produce baseline results against which to compare in later experiments, the data sets described in Table 3.2 were parsed with the 0902 release version of the English Resource Grammar (ERG), using the PET parser configured[2] to produce only one analysis and to use the subsumption-based ambiguity packing of Oepen and Carroll (2000) and the selective unpacking of Zhang *et al.* (2007b). The time limit was set to a maximum of 60 seconds and the supplied *jhpstg.mem* parse ranking model was used. No external information was used for unknown word handling in this baseline experiment. Table 3.3 shows the coverage and efficiency for each set. In this instance, efficiency is shown as the average total time and memory usage per item, as reported by the [incr tsdb()] system. The time includes parsing time only and ignores grammar and model loading time. Both measurements ignore items which failed because of unknown words, but include those items that were not complete within the 60-second time limit. Coverage is the percentage of items which received a parse. For comparison, gold coverage (the *Parsed* column from Table 3.2) is also shown. The procedure for creating these results differed in two ways from that used in creating the gold standard: the gold set was parsed allowing up to 500 analyses per sentence[3] to give the annotators something to select from; and there was some unknown

[2]options: `-nsolutions=1 -packing -timeout=60`
[3]option: `-nsolutions=500`

Test Set	Raw Coverage	Gold Coverage	Unknown Words	Timed Out	Seconds per Item	MB per Item
jh5	95.5%	95.3%	—	0.2%	1.69	99.83
jhpstg$_t$	93.7%	96.3%	2.7%	0.2%	0.97	82.74
ws02	56.8%	90.2%	37.7%	0.5%	2.00	128.45
cb	57.1%	84.4%	33.8%	1.8%	3.85	187.47

Table 3.3: Baseline coverage and efficiency. Raw coverage is the percentage of items that received a parse and the coverage from the gold treebanks is provided for comparison. The percentages of sentences that failed to parse because of unknown words or because parsing had not completed within the 60-second time limit are shown as Unknown Words and Timed Out respectively. Average time and memory are calculated only for those sentences that do not fail instantly due to unknown words and are given in the last two columns.

word handling utilised when the gold set was created. As later experiments will only evaluate the top parse, limiting the parse to one analysis now will allow better comparisons and is a more realistic application scenario. The percentage of parses that failed due to a lack of unknown word handling is shown in column *Unknown Words* and those that timed out (took longer than 60 seconds to parse) in column *Timed Out*.

The first obvious point to take from these results is the major issue that unknown words can cause in unseen text. In both of the web data based test sets parsing fails instantly for over 30% of items because of unknown words. Chapter 5 will discuss unknown word handling mechanisms and to what extent they solve this problem. In terms of efficiency, both time and space show a similar picture: the shorter items from the tourism data are quicker to parse and use less memory, the longer items from *cb* take the most time and memory (and incidently have the highest percentage of timeouts). It is not a direct relationship however and Chapter 6 examines another factor that affects parser efficiency. The parsing speeds shown here, while not unusable, are still perhaps a bit slow for large scale parsing and Chapter 6 reports some experiments designed to improve that aspect of parsing.

3.2.3 Accuracy

The DELPH-IN grammars are often known as precision grammars, but many recent experiments on improving the PET parser or the grammars have focussed on coverage as the primary evaluation. Those that do report precision vary between so-called 'oracle' precision and top-1 precision. Oracle precision evaluates the proportion of sentences that, having produced at least one parse, produces a correct parse. If the parser has been limited to a maximum number of parses, it is possible that the correct parse might be available to the grammar, but not found. Still, by setting the maximum number of parses high enough (the tradition is 500), the effect of parse ranking is reduced, and oracle precision might be thought to evaluate the grammar only, discounting the effects of the parser and parse ranking. Top-1 precision measures the proportion of parsed sentences that have the correct parse ranked highest and so evaluates the grammar and parse ranking model as a complete system. Top-1 precision is a more realistic evaluation where the goal is use of the parser in an application, since an application will generally not have the luxury of a human hand-picking the best parse from those returned. Oracle precision is appropriate to evaluate techniques that affect the grammar (in most cases, more specifically, the lexicon). Importantly, in both cases, the correct parse is the parse that matches the gold standard exactly in every way. Table 3.4 shows oracle precision, top-1 precision and mean reciprocal rank (MRR) for each of the five test sets, where reciprocal rank is the reciprocal of the rank of the analysis that exactly matches the gold standard, or zero if there is no such parse in the top 500. The parser was run with the same configuration as before, except that the top 500 best parses were recorded in order to calculate oracle precision and MRR.

The oracle results might be considered an upper bound on the performance of the parse ranking model. The performance on most data sets is quite high, but there are still a percentage of sentences in each set for which the grammar currently does not have a completely correct analysis, even though some sort of analysis was produced. The top-1 precision assesses the parse ranking model alongside the grammar. As expected, the best performance comes on the *jh5* data set that actually formed part of the training data. The other results were as expected, with those for the in-domain set *jhpstg*$_t$ better than the out-of-domain sets. Here the MRR appears to be simply a more lenient version of top-1 precision but,

Test set	Oracle precision	Top-1 precision	MRR
jh5	0.934	0.656	0.742
jhpstg$_t$	0.948	0.433	0.536
ws02	0.858	0.388	0.470
cb	0.817	0.204	0.295

Table 3.4: Baseline precision. Oracle precision is the proportion of sentences that, having received at least one parse, had a correct parse within the top 500 parses. Top-1 precision is the proportion of parsed sentences that had the correct parse ranked highest and MRR is the mean of the reciprocal rank of parsed sentences, where the reciprocal rank is the reciprocal of the rank of the parse that exactly matches the gold standard, or zero if no such parse is in the top 500 analyses.

where the parse ranking model is the main component being evaluated, it provides a broader picture of any effects of changing the model since it measures not only improvements in the percentage of correct parses ranked on top, but also when the correct parse moves up in the rankings, even if not as far as the top. Of course, if the aim is to focus solely on the effect of the parse ranking model, it might make sense to focus only on that subset of data for which we know a gold analysis exists. In the current configuration, that would have the effect of making the upper bound (oracle precision) 1, since the conditions are the same as those that produced the gold standard, except for the lack of unknown word handling which may affect coverage but not precision.

Our stated aim for evaluation, however, is to assess suitability for a task, which indicates that we should evaluate the parser, grammar and parse ranking model as a complete system, and focus on top-1 evaluation. The problem is that the top-1 precision figures here only give half the picture. By making a binary correct or incorrect decision about the top tree, we ignore the fact that some 'wrong' trees are better than others. Particularly given the detail produced by DELPH-IN HPSG grammars like the ERG, in some cases only subtle distinctions or arbitrary decisions separate some analyses. If the purpose of evaluation is to judge the appropriateness of this grammar and parser for a particular application, it makes more sense to ask *How good is the top result?*, rather than, *Is the top result the very best analysis the grammar is capable of?*. Even when the goal of evaluation is to drive grammar internal improvement, rather than anything application specific, it helps to have a more detailed picture of what the grammar is getting right or wrong, alongside the exact match results. This motivates the experiments in granular evaluation in the next section.

3.2.4 Elementary Dependency Match: A New Granular Evaluation

With a few exceptions, parsing results in the wider community are overwhelmingly not given as functions of exact sentence match. The standard metric in the PCFG parsing community, as discussed in Section 3.1.1, is PARSEVAL, a metric based on counting matching brackets that mark phrase boundaries. In dependency parsing (labelled and unlabelled) triples of <*head,dependency,modifier*> are the elements that are counted. While the DELPH-IN HPSG parser does produce trees as part of its analysis (derivation trees as well as the compacted phrase structure trees used in treebanking), our hypothetical application requires semantics, and so it makes sense to evaluate this semantics, rather than the steps used to obtain it. While the canonical representation of these semantics is an MRS structure (as described in Section 2.1.1), it is possible to present most of the information (excluding scopal information) in the MRS in the form of triples, abstractly similar to those in dependency parsing.

These triples are formed from what Oepen and Lønning (2006) have defined as Elementary Dependencies. In their work, details from these dependencies were used as discriminants denoting individual differences between different analyses, and annotators made binary decisions regarding the correctness of these discriminants to select the correct analysis from amongst a set of possible analyses of a sentence. Oepen and Lønning described three different forms of discriminant:

(a) $relation_i$

(b) $relation_i$ $role_j$ $relation_k$

(c) $relation_i$ $property_j$ $value_j$

In these forms, *relation* is the predicate name of an elementary predication (EP) from the MRS, *role* is an argument type such as ARG1, *property* refers to an attribute such as TENSE or GEND and *value* is an appropriate instantiation for the respective property. For evaluation purposes, we are also interested in individual differences between analyses, but in our case, we want to count them and so small alterations are required to the above definitions. In our Elementary Dependency Match (EDM) metric, we use the three types of discriminants listed above, however, in those forms an error in predicate name is propagated to each discriminant bearing that relation. To counter that issue, instead of using the predicate name to identify a relation, we use the character span associated with the relation's EP, which links to the span of the input text that triggered the creation of that predicate. To relate the actual predicate relation to the character span, we introduce a meta-relation NAME. Now we have three types of triples:

NAMES:	$span_i$	NAME	$relation_i$
ARGS:	$span_i$	$role_j$	$span_k$
PROPS:	$span_i$	$property_j$	$value_j$

Figure 3.1 shows an example of the triples for the gold analysis of *A declaration associates a variable name with a datatype.* with the ARGS highlighted. These triples represent the core of the semantic information contained in the analysis, except for scopal relations and they provide an easy way to see how an analysis of a sentence differs from the gold analysis. Figure 3.2 shows the verbose output of comparing an analysis against the gold standard, using the `diff` convention of marking missing lines (or triples) with < and extra lines with >.

In this representation of the differences between analyses, it can be seen that in the top-ranked parse, *variable* was mis-analysed as an adjective, rather than a noun. This meant first that the associated predicate was assigned the properties of an adjective, and secondly that *variable name* was considered an adjective-noun construction, rather than a noun compound. The numeric results are calculated over all EDM triples, as well as separately over the three different types defined earlier. In this example, all four sets of numbers are similar, but different sorts of errors can have greater impact on one triple type over another, leading to wider variation between the sets of numbers.

An EDM evaluation of our four data sets was carried out using the same parser options as those used for the coverage and efficiency evaluations[4] and hence only evaluating the top parse as selected by the statistical model. Since it is not possible to evaluate an analysis for which there is no gold standard, the precision, recall and f-score are calculated across

[4]options: `-nsolutions=1 -packing -timeout=60`

“a”	<0:0>	NAME	_a_q	
“a”	**<0:0>**	**ARG0**	**<2:12>**	**“declaration”**
“declaration”	<2:12>	NAME	_declaration_n_of	
“declaration”	<2:12>	PERS	3	
“declaration”	<2:12>	NUM	sg	
“associates”	<14:23>	NAME	_associate_v_with	
“associates”	**<14:23>**	**ARG1**	**<2:12>**	**“declaration”**
“associates”	**<14:23>**	**ARG2**	**<40:43>**	**“name”**
“associates”	**<14:23>**	**ARG3**	**<54:64>**	**“datatype.”**
“associates”	<14:23>	SF	prop	
“associates”	<14:23>	TENSE	pres	
“associates”	<14:23>	MOOD	indicative	
“associates”	<14:23>	PROG	-	
“associates”	<14:23>	PERF	-	
“a”	<25:25>	NAME	_a_q	
“a”	**<25:25>**	**ARG0**	**<40:43>**	**“name”**
“variable name”	<29:43>	NAME	compound	
“variable name”	<29:43>	NAME	udef_q	
“variable name”	**<29:43>**	**ARG0**	**<29:36>**	**“variable”**
“variable name”	**<29:43>**	**ARG1**	**<40:43>**	**“name”**
“variable name”	**<29:43>**	**ARG2**	**<29:36>**	**“variable”**
“variable name”	<29:43>	SF	prop	
“variable name”	<29:43>	TENSE	untensed	
“variable name”	<29:43>	MOOD	indicative	
“variable name”	<29:43>	PROG	-	
“variable name”	<29:43>	PERF	-	
“variable”	<29:36>	NAME	_variable_n_1	
“variable”	<29:36>	IND	+	
“name”	<40:43>	NAME	_name_n_of	
“name”	<40:43>	PERS	3	
“name”	<40:43>	NUM	sg	
“name”	<40:43>	IND	+	
“a”	<50:50>	NAME	_a_q	
“a”	**<50:50>**	**ARG0**	**<54:64>**	**“datatype.”**
“datatype.”	<54:64>	NAME	_datatype_nn	
“datatype.”	<54:64>	PERS	3	
“datatype.”	<54:64>	NUM	sg	

Figure 3.1: EDM triples for the gold analysis of *A declaration associates a variable name with a datatype.* The input text span associated with the character spans is shown for illustrative purposes in the first and last columns.

```
<        “variable”  <29:36>  NAME   _variable_n_1
>        “variable”  <29:36>  NAME   _variable_a_1

<        “variable”  <29:36>   IND   +
>        “variable”  <29:36>   SF    prop
>        “variable”  <29:36>  TENSE  untensed
>        “variable”  <29:36>  MOOD   indicative

<   “variable name”  <29:43>  NAME   compound
<   “variable name”  <29:43>  NAME   udef_q
<   “variable name”  <29:43>  ARG0   <29:36>     “variable”
<   “variable name”  <29:43>  ARG1   <40:43>     “name”
<   “variable name”  <29:43>  ARG2   <29:36>     “variable”
<   “variable name”  <29:43>   SF    prop
<   “variable name”  <29:43>  TENSE  untensed
<   “variable name”  <29:43>  MOOD   indicative
<   “variable name”  <29:43>  PROG   -
<   “variable name”  <29:43>  PERF   -
>        “variable”  <29:36>  ARG1   <40:43>     “name”

ALL    Precision: 0.833  Recall: 0.676  F-score: 0.746
NAMES  Precision: 0.875  Recall: 0.700  F-score: 0.778
ARGS   Precision: 0.857  Recall: 0.667  F-score: 0.750
PROPS  Precision: 0.800  Recall: 0.667  F-score: 0.727
```

Figure 3.2: Verbose output of a comparison of EDM triples from the gold and top-1 analyses of *A declaration associates a variable name with a datatype.* Numeric results are given for all triples, as well as separately for the NAMES, ARGS and PROPS triple types. Lines headed by < depict triples missing in the top-1 analysis, while those headed by > show triples that are incorrectly included in the top-1.

	EDM evaluation			Exact match evaluation		
Test set	Precision	Recall	F-score	Precision	Recall	F-score
jh5	0.965	0.965	0.965	0.702	0.702	0.702
jhpstg$_t$	0.921	0.903	0.912	0.456	0.447	0.451
ws02	0.888	0.520	0.656	0.452	0.289	0.352
cb	0.884	0.611	0.723	0.250	0.177	0.207

Table 3.5: EDM evaluation over all triple types measured across all items for which a gold standard exists. Exact match evaluations are shown for comparison. Exact match precision is the same as top-1 precision from Table 3.4, except measured only over those sentences with a gold standard. Exact match recall and precision have the related definitions.

only those test items for which a gold analysis has been recorded. These results are shown in Table 3.5 along with exact match precision, recall and f-score calculated over the same test items for comparison. This exact match precision is the same as the top-1 precision given in Table 3.4, except that it is only calculated over those sentences for which a gold standard exists, and so the numbers are slightly higher. The associated recall and f-score figures are included to allow direct comparison between the granular and exact match evaluation metrics. One noticeable variation between the two occurs when looking at the relative differences between performance on the *ws02* and *cb* data sets. While recall on *cb* according to exact match is lower, showing that the best parse is not often ranked top, recall according to EDM is higher, indicating that a higher proportion of relations are correct in *cb* than in *ws02*.

Table 3.6 reports the best published granular evaluations of other popular parsers. None of these results are directly comparable, since they use different evaluation schemes and different test sets, but they give an overview of state-of-the-art parsing. While compiling these results, many of the problems involved in comparing across parsers came up: some system use gold standard POS tags (hardly a realistic scenario) and others don't; many use a subset of the Penn Treebank Section 23, but that subset changes; some ignore punctuation and others count it; some results are calculated over just those sentences parsed, while others use all sentences. In some cases, there was not enough information given to determine these variables for each set of results.

Clark and Curran (2007b) and Miyao and Tsujii (2008) use probably the closest evaluation schemes to the one described here, although

	Data Set	Precision	Recall	F-score
Collins and Koo (2005)	PTB §23 (<100 words)	89.6	89.9	89.7
Charniak (2000)	PTB §23 (<100 words)	89.6	89.5	89.5
McClosky *et al.* (2006a)	PTB §23	n/a	n/a	92.1
Kaplan *et al.* (2004)	PARC 700 DepBank	79.4	79.8	79.6
Briscoe and Carroll (2006)	GR DepBank	81.5	78.1	79.7
Clark and Curran (2007b)	CCGBank §23	86.17	84.74	85.45
Miyao and Tsujii (2008)	PTB §23 (<100 words)	86.47	85.83	86.15

Table 3.6: State-of-the-art English parsing performance

neither evaluate the property values as well as the relations. In order to be more comparable with this work, as well as because relations are considered more important than properties in many applications, we define an EDM_{NA} evaluation that combines the NAMES and ARGS EDM triples. Table 3.7 shows precision, recall and f-score using this EDM_{NA} evaluation. Sentence accuracy figures are also given, where a sentence is considered correct if all EDM_{NA} triples in that sentence are correctly identified. Clark and Curran report a sentence precision of 32.92% and Miyao and Tsujii a sentence recall of 33.8%. Our sentence accuracy for the in-domain *jhpstg*$_t$ data set is quite a bit higher, but this is probably due to the relatively short average sentence length (12.87 words for this subset, compared to 22.23 for Miyao and Tsujii). The sentence accuracy results for *ws02*, which is out-of-domain for the parse ranking model, are very good, considering the low coverage on this test set — almost half of the sentences that parsed had all ARGS and NAMES correct. For the *cb* data set, the sentence results are much lower than the EDM_{NA} evaluation, echoing the difference we saw in Table 3.5 between EDM and exact match, since the longer sentences make it more likely that there will be at least one error in the analysis. Looking at the EDM_{NA} results for the out-of-domain test tests, we see they both suffer from a low recall caused by low parse coverage. Chapter 5 will attempt to address this problem.

This EDM evaluation scheme has not been previously used in evaluating the accuracy of DELPH-IN grammars and parsers, but it has attractive features that make it a promising candidate for the as-yet undefined

	EDM_{NA} evaluation			Sentence Accuracy		
Test Set	Precision	Recall	F-score	Precision	Recall	F-score
jh5	0.952	0.952	0.952	0.729	0.729	0.729
jhpstg$_t$	0.899	0.881	0.890	0.497	0.487	0.492
ws02	0.857	0.501	0.632	0.469	0.304	0.369
cb	0.846	0.585	0.692	0.268	0.193	0.224

Table 3.7: EDM_{NA} evaluation, which evaluates both NAMES and ARGS triple types, but not PROPS. Sentence accuracy figures are calculated based on a sentence being correct if all EDM_{NA} triples in that sentence are correct.

internal granular evaluation standard for these parsers. It measures the same sort of atomic units that other deep parsers have used for evaluation, it evaluates semantics directly and it is readily understandable to application developers as Figure 3.2 should illustrate. The results can also be broken up by predicate type in order to get a closer look at where the parser is going wrong, which makes it a useful evaluation scheme for focussing parser development, but also for evaluating just those relations that are useful to an application. For example, the EDM_{NA} evaluation may be the most suitable for an application, but it would also be possible to only evaluate ARG1 and ARG2 relations, as the most important. Another variation could weight the relations differently. The EDM_{NA} evaluation is the equivalent of setting all PROPS type triples to a weight of zero, and all other types to weight one, but a more nuanced weighting scheme is also possible. Evaluation profiles describing these weights, in the style of the parameter files for EvalB (Sekine and Collins 1997) would be one way to standardise the parameters of this evaluation scheme, making sure that the evaluation profile was available for any published set of results to aid later comparisons.

3.3 Conclusion

This chapter attempted to give a clear definition of parser evaluation, emphasising how the goals of parsing and of evaluation can affect the ideal evaluation method. Different evaluation criteria, such as efficiency and robustness were discussed, and then an overview was given of the attempts of the parsing community to define parse accuracy evaluation metrics. This overview summarised some of the proposed schemes based on either phrase structure boundaries or dependency relations, and the case made for the latter over the former. Section 3.1.3 focussed on the bugbear of cross-framework parser evaluation, inter-representation mapping, showing the difficulty of the task, even within frameworks. Some of the research summarised here raised the question of whether there is any validity in comparing between mapped results when the mapping process can have a greater effect on the results than any difference in parsing performance. Section 3.1.4 discussed how, even among relation-based evaluation schemes, the granularity of the relations can have a big impact on a final evaluation score. Again, this problem arises because of unstated disagreements over the goals of both parsing and evaluation. The aim of the second half of this chapter was to look at how the PET parser has been evaluated before and, using the ideas developed through the first half, define the evaluation metrics that will be used in this thesis. The first lesson learnt was to define the goals of parsing and evaluation. Throughout this work, we will assume that the goal of parsing is to produce a semantic analysis to be used in an application, where the semantic analysis should describe all the semantics that is encoded syntactically. Specifically, the application requires the answers to *Who did what to whom, when, where and how?* The evaluation goals therefore are to assess the parser suitability for this task, and to compare between different parser configurations, given this task. Evaluation metrics previously used for this parser were examined and it was found that these metrics were suitable where the goal was grammar development or, in other cases, evaluating the parse ranking model. For the current parsing and evaluation goals, however, a granular accuracy evaluation metric seemed more appropriate, and Section 3.2.4 described the EDM evaluation scheme that we will use, including details of the level of granularity and why they were selected. The results in Tables 3.3, 3.5 and 3.7 will provide the baseline for experiments in subsequent chapters.

4 The Supertagger

Many different types of lexical statistics can be learnt from annotated and unannotated text. Word frequencies, part of speech categories and named entity information are all frequently used to inform natural language processing systems, and tools are readily available to produce this information, at least for English. Supertags, as described in Section 2.3, are also an effective source of lexical information. This effectiveness comes, in large part, from the close integration with the linguistic formalism used in parsing, which means that off-the-shelf supertaggers complete with model are not so easily come by. This chapter will detail experiments designed to find an appropriate supertagger for using with the PET HPSG parser, where *appropriate* might vary depending on how the supertagger is to be used. Later chapters will investigate not only different ways of integrating the supertagger into the parser for search space restriction, but also how it can be used for unknown word handling and parse ranking.

4.1 The Tags

The first step is to define the form of the supertag. For both CCG and LTAG, it appears this step was straightforward, since the CCG categories and the LTAG elementary trees are the elements stored in the lexicon. In HPSG, this is not as clear-cut. Section 2.3.3 summarised a few of the different types of supertag that have been used in HPSG parsing. This variation is a matter both of implementation as well as formalism. The information contained in a HPSG sign of type *word* comes from different sources, and how much is explicitly encoded in the lexicon varies according to the grammar implementation. Figure 4.1 shows an example of a lexical entry from the DELPH-IN ERG we used for evaluation in Chapter 3.

Most of the information from this lexical entry is specific to the form and meaning of the associated word: the STEM and PHON.ONSET relate to the form, while the KEYREL.PRED is the semantic predicate that will contribute to the semantic analysis. The –COMPKEY feature is not present for every lexical entry and encodes the fact that this entry selects for a preposition of form *of* in its complement. The more general category information associated with this lexical entry is represented by the **lexical type** *v_pp_e_le*, which generalises subcategorisation information.

think_of := v_pp_e_le &

$$\begin{bmatrix} \text{STEM} & \langle \text{“think”} \rangle, \\ \text{SYNSEM} & \begin{bmatrix} \text{LKEYS} & \begin{bmatrix} \text{–COMPKEY} & \text{\_of\_p\_sel\_rel} \\ \text{KEYREL.PRED} & \text{“\_think\_v\_of\_rel”} \end{bmatrix} \\ \text{PHON.ONSET} & \text{con} \end{bmatrix} \end{bmatrix}$$

Figure 4.1: Example lexicon entry from the ERG.

`v_pp_e_le`

$$\begin{bmatrix} \text{SYNSEM} & \begin{bmatrix} \text{LOCAL} & \begin{bmatrix} \text{CAT} & \begin{bmatrix} \text{HEAD} & \text{verb} \\ \text{VAL} & \begin{bmatrix} \text{SUBJ} & \left\langle \begin{bmatrix} \text{LOCAL} & \begin{bmatrix} \text{CAT} & \text{nomp\_cat} \end{bmatrix} \end{bmatrix} \right\rangle \\ \text{SPR} & \langle \rangle \\ \text{COMP} & \left\langle \begin{bmatrix} \text{LOCAL} & \begin{bmatrix} \text{CAT} & \text{basic\_pp} \end{bmatrix} \\ \text{–MIN} & \boxed{1}\ \text{selected\_rel} \end{bmatrix} \right\rangle \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \text{LKEYS} & \begin{bmatrix} \text{–COMPKEY} & \boxed{1} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Figure 4.2: Lexical type `v_pp_e_le`, simplified

To a rough approximation, this lexical type represents the same sort of information encoded in a CCG category or an LTAG elementary tree. There are two main differences however. First, this lexical type is not an atomic value, but represents a large feature structure, shown in simplified form in Figure 4.2. While the names of the lexical types have been chosen to reflect the feature structure details, they do not completely describe the lexical type. The other major difference between HPSG lexical types and supertags from other formalisms is that the lexical type describes the category of a lexical entry, not a word. In concrete terms, this means that *think*, *thinks* and *thought* would all trigger this lexical entry and hence this lexical type. Information from lexical rules, triggered by morphology, is added to that of the lexical entry to produce the lexical item used in parsing. The terms lexical entry, lexical type and lexical item are easily confused, but do refer to different types of entity. Figure 4.3 contains definitions for each of these terms.

The decision as to the most appropriate supertag for HPSG should

Lexical Entry Element stored in the lexicon. Made up of, at least, an identifier, a lexical type, a stem (used as a look up key in parsing), a semantic predicate (look up key during generation) and phonetic information.

Lexical Type A feature structure containing category information for the lexical entry. Describes at least head type and valence information.

Lexical Item A *word* sign used in parsing. A combination of the information from a lexical entry and from one or more lexical rules.

Figure 4.3: Definitions of the related, but separate terms **lexical entry**, **lexical type** and **lexical item**.

take into consideration two possibly competing requirements. Previous research has discussed these two requirements as internal and external qualities (e.g. Dickinson and Jochim 2008) or computational and linguistic qualities (Leech 1997), looking specifically at the linguistic distinctions. For this work, however we take an entirely practical point of view and think of these requirements as predictability, i.e. the tags can be accurately predicted from local surface information; and usefulness, where useful means 'improves parser performance'. While we expect that a *useful* tag set will in fact make linguistically meaningful distinctions, which distinctions are useful will depend on the make up of the parser and grammar. Looking again at the lexical entry then, the non-word specific information is contained in the lexical type and the selectional relations. The more information that can be predicted, the more useful the supertags will be for their particular purpose. However, in order to be effective, they also need to be predictable. This chapter focusses on measuring this predictability, with later chapters determining the usefulness of the tags. The granularity that gives the best trade-off between useful and predictable is something that will be determined empirically. To this end, four tag granularities are defined.

The most fine-grained tag type is a concatenation of the lexical type and any selectional relation in a lexical entry, which will be referred to as **letype+sel**. A natural coarsening of this form is to use the lexical type (**letype**) on its own, as Blunsom (2007) did in his experiments, described in Chapter 2. These lexical types are still quite specific, and so two further generalisations are defined. As mentioned above, the name of the lexical type does not reflect all properties of the type, but the types were deliberately named according to a fixed pattern as a mnemonic for people using the grammar. This pattern is broken up by underscores (_) and is of the form <part-of-speech>_<subcategorisation>_<description>_le. The part-of-speech is one of a small set of broad categories such as **v**:verb or **aj**:adjective. The subcategorisation gives the types of the arguments, including marking some arguments as optional, hence **np-np** would be the subcategorisation of a standard ditransitive verb, where **pp*-cp** represents an optional prepositional phrase and an obligatory complementiser. The description field can further specify the lexical type, indicating, for instance, a particular complementiser or an idiomatic use. Not all lexical types have a description field. According to this break down, we can see that the lexical type *v_pp_e_le* represents a verb with a prepositional phrase argument, where the preposition is semantically empty. Using this fixed pattern, we define two further supertag forms. The **subcat** form consists of the first two fields of the lexical type and, by leaving off the often semantic distinctions given in the description field, is quite close to LTAG elementary trees. The **pos** tag is the least fine-grained tag and is made up of only the first (part-of-speech) field of the lexical type. According to the earlier definition — that supertags embody richer information than POS tags, particularly dependency information — these last tags would not even be called supertags. They are included as a baseline to determine whether the power of supertags is needed in all cases. Since these **pos** tags are derived from the lexical type, they match the distinctions made in the DELPH-IN grammars, something that the Penn Treebank tags do not always do.

All the above supertag forms relate to the lexical *entry.* Another variation, which would be closer to the effective distinctions made in CCG and LTAG, would be to tag the lexical *item.* Ninomiya *et al.* (2006) did this by concatenating the lexical rules and the lexical types of the ENJU grammar, as in **D<N.3sg>_lxm-noun_adjective_rule**, where **D<N.3sg>_lxm** represents one of their syntactic lexeme categories (lexical types) and **noun_adjective_rule**, an inflectional rule. The analogous method for the DELPH-IN grammars would be to add the morphological rules to each level of granularity, hence giving 8 tag forms which are summarised in Table 4.1.

One common consequence of making fine distinctions is that many of the tags will not be frequently seen; this pattern is obvious when looking

Tag Type	Example	Number seen in training
letype+sel+morph	v_pp_e_le +_of_p_sel_rel +third_sg_fin_verb_orule	1217
letype+sel	v_pp_e_le+_of_p_sel_rel	803
letype+morph	v_pp_e_le+third_sg_fin_verb_orule	996
letype	v_pp_e_le	676
subcat+morph	v_pp+third_sg_fin_verb_orule	254
subcat	v_pp	110
pos+morph	v+third_sg_fin_verb_orule	36
pos	v	13

Table 4.1: Supertag types, with examples. There are four granularities based on the syntactic information represented by the lexical type and selectional relations. In addition, each granularity can be used with and without morphological information. The numbers show how many distinct values were seen for each type in the basic training data set that will be described in Section 4.2.

at the frequency distributions of the tags in the basic training data set, described in Section 4.2. Examining the distribution of the **pos** set, it is fairly balanced, with nouns being about twice as popular as the next most frequent tag, but the next five tags having similar frequencies, and there is only one rare tag. As the tag sets get more fine-grained, this is no longer the case. None of the tag sets show a strictly Zipfian distribution, since many of the most common tags in each distribution show similar frequencies, but all the fine-grained tag sets have long tails, with many tags being seen infrequently in the training data. Table 4.2 shows the percentages of infrequent tag values of each type, broken down into the percentage seen less than 100 times, less than 10 times and only once in the 157,920 token training set.

4.2 The Training Data

All of the training data comes from the dynamic treebanks released with the 0902 release of the ERG. The total token size of the training set is 157,920 tokens, with almost 94% of that being tourism data from the LOGON project (Oepen *et al.* 2004a). The remainder is made up of test

Tag Type	<100 times	<10 times	only once
letype+sel+morph	84.0%	54.8%	17.7%
letype+sel	79.0%	44.2%	11.2%
letype+morph	80.8%	46.0%	12.2%
letype	75.1%	38.0%	8.3%
subcat+morph	72.4%	41.3%	11.4%
subcat	56.4%	32.7%	7.3%
pos+morph	19.4%	8.3%	0.0%
pos	15.4%	7.7%	0.0%

Table 4.2: The long tail: percentages of infrequent tags in each tag set, broken down into the percentage seen less than 100 times, less than 10 times and only once in the 157,920 token training set (described in Section 4.2).

suites designed to test particular syntactic and semantic phenomena. Table 4.3 lists the treebanks in the training set, along with their vital statistics. These statistics represent only those items which have a gold analysis and hence can be used for training.

4.2.1 Extracting the data

As described in Section 3.2.1, these treebanks have been constructed by parsing the data and then having an annotator select the correct parse out of the top 500 produced by the parser. Only those parses annotated as completely correct have been used in training. The training data was extracted from gold derivation trees, but this was not a completely straightforward process. As discussed in Section 2.4, the ERG does not natively use Penn Treebank tokenisation. Figure 4.4 shows an example derivation tree for the sentence *The natural environment was favorable, at least.* Here we can see some of the possible issues involving mismatches between the expected input and the output tokenisation. In the token *favorable,*, we see the comma attached to the word, one of the key differences between Penn Treebank tokenisation and that of the ERG. Even if it were decided to use the ERG native tokenisation as input to the supertagger, that would not solve the problem displayed in the last token, *at least.* In this case, the grammar writer has lexicalised the multiword expression *at least*, since the meaning was thought to be non-

Treebank	Items	Tokens	Tokens per Item	Comment
csli	880	6660	7.57	CSLI (LinGO) Test Suite
fracas	325	2843	8.75	FraCaS Semantics Test Suite
hike	328	4726	14.41	Tourism Data
jh0	233	4904	21.05	Tourism Data
jh1	1250	18737	14.99	Tourism Data
jh2	1130	17159	15.18	Tourism Data
jh3	1302	18701	14.36	Tourism Data
jh4	1456	20917	14.37	Tourism Data
jh5	414	5540	13.38	Tourism Data
mrs	107	594	5.55	DELPH-IN MRS Test Suite
ps	874	12896	14.76	Tourism Data
rondane	1108	17251	15.57	Tourism Data
tg1	877	13202	15.05	Tourism Data
tg2	891	13790	15.48	Tourism Data
Total	**11175**	**157920**	**14.13**	

Table 4.3: The treebanks used for training the supertaggers. The treebanks come from the 0902 release of the ERG. The numbers describe only those items which have a gold analysis.

compositional and hence, in this analysis, two ERG native tokens have been joined to form one leaf node of the derivation tree. These lexicalised multiword expressions are fairly common in the grammar, and so using the native ERG tokenisation as input to the supertagger would not make extracting the token-to-tag matches straightforward anyway. As such, it was decided to use Penn Treebank tokenisation in the supertagger input, making it easier to integrate with standard part-of-speech taggers.

A pre-processing step tokenised the raw input according to Penn Treebank tokenisation rules and then calculated the best alignment between these tokens and the leaves of the derivation tree. To find the tag for each leaf, the pre-terminal lexical entry (e.g., `environment_n1`) was looked up in the lexicon, and the lexical type and any selectional relations were extracted. The morphological information, when it occurred, was found in the tree above the pre-terminal node, for example, the `sing_noun_irule` in Figure 4.4. The different forms of supertag were created from the lexical type, selectional relation information and morphological rule in the eight forms described in the preceding section. All non-punctuation tokens associated with a leaf were given the tag associated with that leaf. This led to the token *at* being assigned an adverbial tag in this training example. No attempt was made to indicate that a token was part of a multiword expression. Since both taggers we use employ a sequence-aware model, it was thought that idiosyncrasies of multiword expressions could be learnt without introducing unnecessary data sparsity. For punctuation tokens that do not have a lexical type, the **letype+sel**, **letype** and **subcat** supertags were the same form: PUNCT_<surface>, e.g., PUNCT_,. The **pos** supertag was simply PUNCT, and the **+morph** information in each case was the punctuation-related morphological rule found in that branch of the tree, e.g., `punct_comma_orule`.

4.3 The Taggers

We tested two off-the-shelf taggers that use different statistical models. The first is the TnT POS tagger, an implementation of the Viterbi algorithm for second order Markov models (Brants 2000b). This tagger has been used for a variety of languages and tag sets and is very fast both to train and to use. The input we used for this tagger was the raw tokens, one per line and the output can be either a single tag, or multiple tags with their appropriate probabilities. TnT has a reported tag accuracy

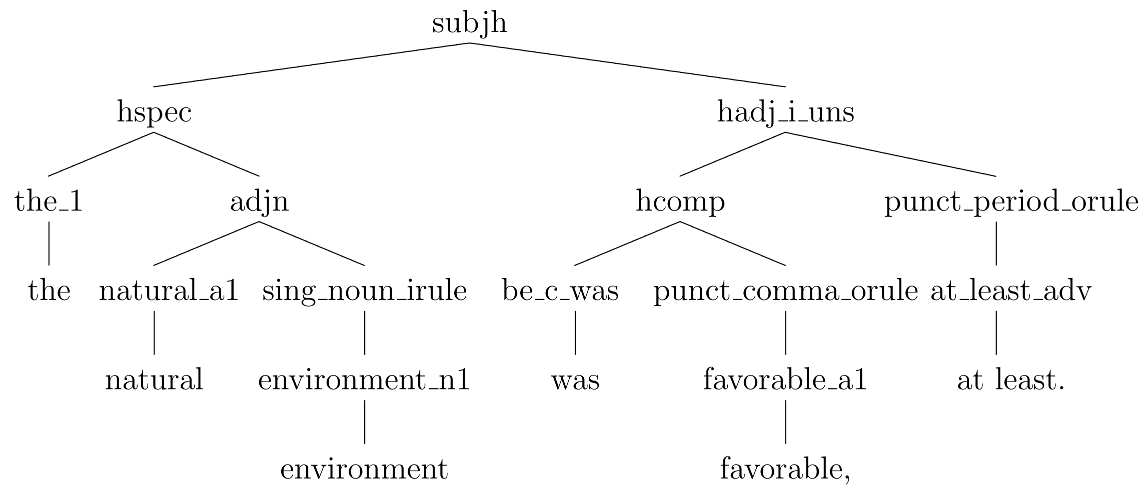


Figure 4.4: Derivation tree for *The natural environment was favorable, at least.*

of 96.7% on Penn Treebank Wall Street Journal text, using the Penn Treebank POS tags, and just over a million tokens of training data.

The state-of-the-art for supertagging is the C&C supertagger, which is, like TnT, freely available (Clark and Curran 2007b). The tagger is based on a Maximum Entropy model, and uses as features words and POS tags from a five word window, as well as the previous two tags assigned. The C&C tagger uses a form of the Viterbi algorithm to calculate the optimal tag sequence and gets a single tag accuracy of 91.5% on Section 00 of CCGBank, using the 425 category tag set, and also using just over a million tokens of training data. As input to this tagger, we use the raw tokens and the POS tag as assigned by TnT, using the Wall Street Journal model released with TnT. As with TnT, output can be single tags, or multiple tags with probabilities.

As a baseline, we use a simple unigram tagger that assigns the tag most frequently seen with a word, if that word was seen in the training data, or else the most frequent tag given the POS tag assigned by TnT. Note that this baseline actually has more input information than the TnT tagger, which does not benefit from the Wall Street Journal POS tag model.

4.4 Results

Each tagger was used to tag three different data sets (fully described in Section 3.2.1), *jhpstg*$_t$ which is from the same tourism domain as the

	Baseline	TnT	C&C
letype+sel+morph	0.8031	**0.9011**	0.8777
letype+sel	0.8305	**0.9143**	0.8898
letype+morph	0.8033	**0.9015**	0.8783
letype	0.8306	**0.9147**	0.8908
subcat+morph	0.8446	**0.9266**	0.9125
subcat	0.8800	**0.9432**	0.9309
pos+morph	0.8901	**0.9510**†	0.9452†
pos	0.9331	**0.9713**†	0.9674†

† marks pairs of results in each row which are not significantly different to each other, statistically

Table 4.4: Single tag accuracy for the $jhpstg_t$ data set

majority of the training data; *ws02*, which is from Wikipedia; and *cb*, a technical essay. Both the latter data sets are completely out of domain, given the training data. We evaluate the accuracy of the top tag, and also two alternative tagging strategies: one that assigns multiple tags depending on the tag probability, and another that selectively assigns a tag, also dependent on the tag probability.

4.4.1 Single Tag Accuracy

The first evaluation looked only at the top tag predicted by each tagger. Tables 4.4, 4.5 and 4.6 show the tag accuracies achieved for each data set. Differences between all results are statistically significant ($p < 0.001$) according to the sign test, except where marked.

While accumulating the results, an interesting difference between the taggers was noticed. In these experiments, we trained models for each level of tag granularity, and initially evaluated only the predictions of the granularity in the model, but then also decided to evaluate all possible tag granularities from each model. That is, when training and predicting, for example, an **letype** level tag, it is also possible to evaluate **pos** and **subcat** tags, but not **letype+sel** or any of the **+morph** type tags. Here we found that the TnT tagger gets better results when trained on more detailed tags, even when the goal is to predict **pos** tags. This echoed results found by the TnT author, Brants (1997) when he looked at different clusterings of the Susanne tag set. In the experiments

	Baseline	TnT	C&C
letype+sel+morph	0.6118	**0.6722**	0.6297
letype+sel	0.6594	**0.7163**	0.6458
letype+morph	0.6118	**0.6727**	0.6298
letype	0.6594†	**0.7169**	0.6460†
subcat+morph	0.6698	**0.7294**†	0.7187†
subcat	0.7953†	**0.8177**	0.7967†
pos+morph	0.7419	0.8060	**0.8274**
pos	0.9047	0.9177	**0.9328**

† marks pairs of results in each row which are not significantly different to each other, statistically

Table 4.5: Single tag accuracy for the *ws02* data set

	Baseline	TnT	C&C
letype+sel+morph	0.6755	**0.7165**	0.6943
letype+sel	0.6751	**0.7453**	0.7024
letype+morph	0.6439	**0.7173**	0.6819
letype	0.6752	**0.7461**	0.7040
subcat+morph	0.6944	**0.7667**†	0.7582†
subcat	0.7651	**0.8188**†	0.8117†
pos+morph	0.8802	0.9100	**0.9306**
pos	0.7761	0.8395	**0.8623**

† marks pairs of results in each row which are not significantly different to each other, statistically

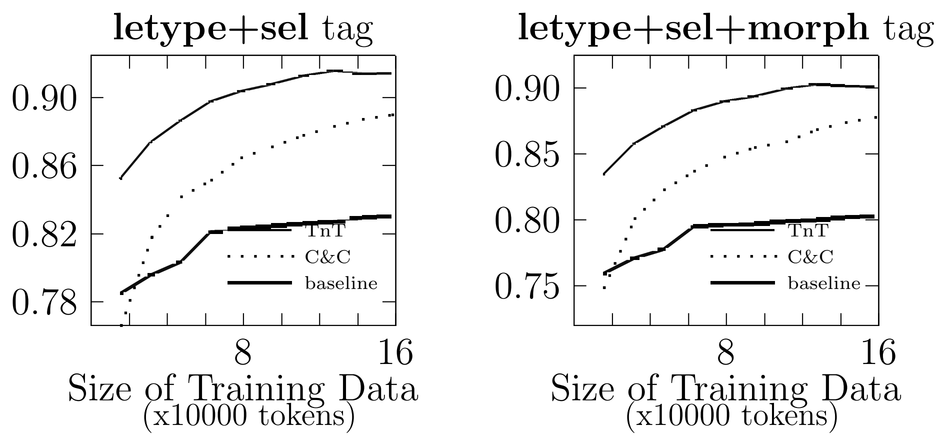
Table 4.6: Single tag accuracy for the *cb* data set

here, there was not, in general, a significant difference between training on **letype+sel+morph** and **letype+morph**, but the accuracy difference between the model trained on **letype+sel+morph** tags and that trained on **pos** tags could be as much as 3% when predicting **pos** tags. Turning then to the C&C tagger, we found that the differences between models were smaller, but that the most accurate model was generally that trained on the exact granularity that was being evaluated, perhaps because the additional POS tag input helped to provide the disambiguation that detailed tags could contribute. The baseline unigram model also followed this trend. When collating the results for the tables then, we used matching training and prediction tag granularities for the C&C and baseline taggers, but those from the **letype+sel+morph** model for all TnT experiments. It may not always be realistic to train a model at this level of detail when data is to be hand annotated, but since we take training data from the parser output, it seemed appropriate to explore the most effective training strategy in each case.

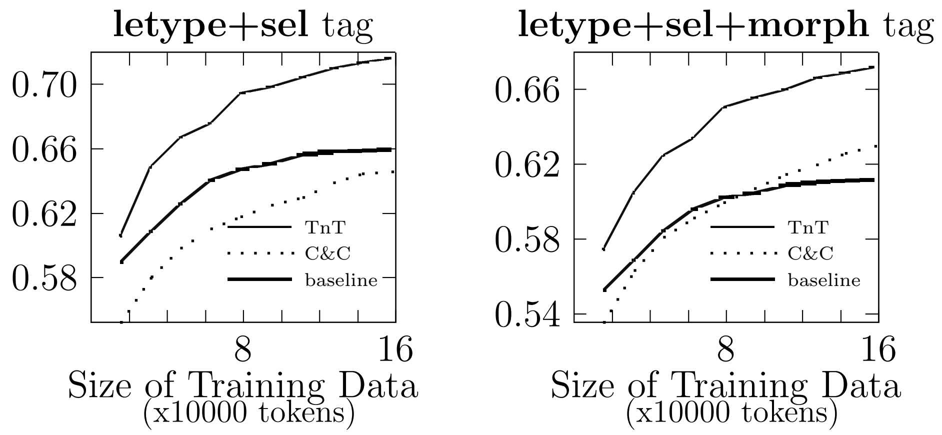
As expected, the best results are on the test set which is closest in domain to the training data. For the most detailed tags the best accuracy on this set is close to that achieved with the C&C supertagger when predicting CCG tags (Clark and Curran 2007b). In that experiment, Clark and Curran used five times the amount of training data we have here, suggesting that the HPSG tag sets, while larger, are actually easier for the tagger to predict.

The best accuracy on the **pos+morph** tag set, closest in size and distinctions made to the Penn Treebank POS tags, is 95.1%, just below state-of-the-art (96%-97%) for POS tagging the Penn Treebank. According to the learning curves given by Brants (2000a), this would be normal performance for predicting the Penn Treebank tags with the amount of training data used in this instance.

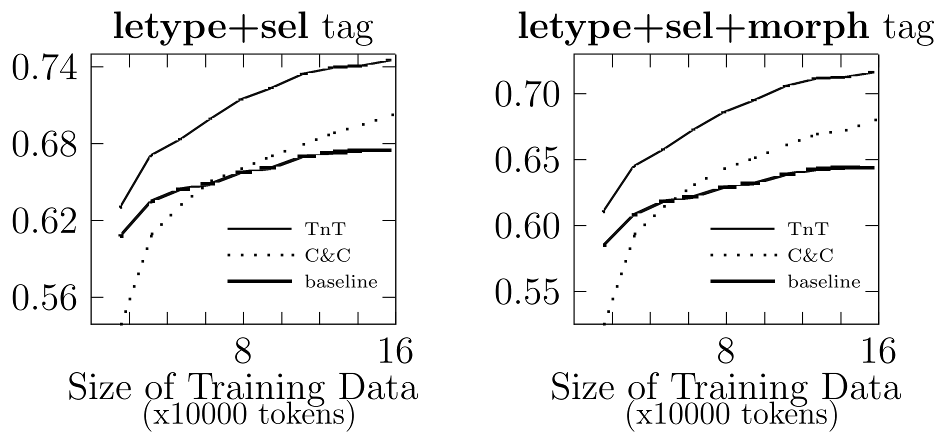
Focussing on individual tagger performance, the simple TnT tagger does better than the Maximum Entropy-based C&C tagger, although this difference narrows as the tags get coarser, and for the **pos** and **pos+morph** tags the performance difference is either insignificant, or slanting in C&C's favour. This suggests that there is not enough training data to learn an accurate model for the detailed tags. To get a better picture of how the amount of training data affects each tagger, we measured tag accuracy for models trained on varying amounts of data, increasing in 10% intervals. Figures 4.5, 4.6, 4.7 and 4.8 show the learning curves for the different tag granularities.



(a) Tag accuracy over the *jhpstg*$_t$ data set

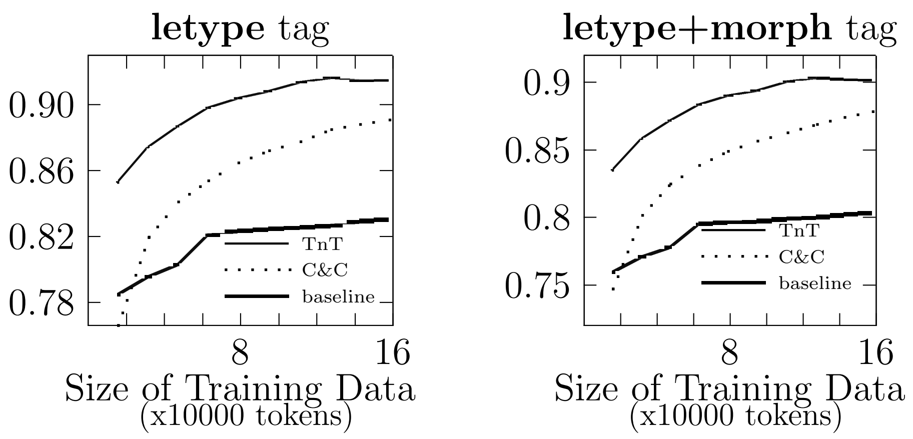


(b) Tag accuracy over the *ws02* data set

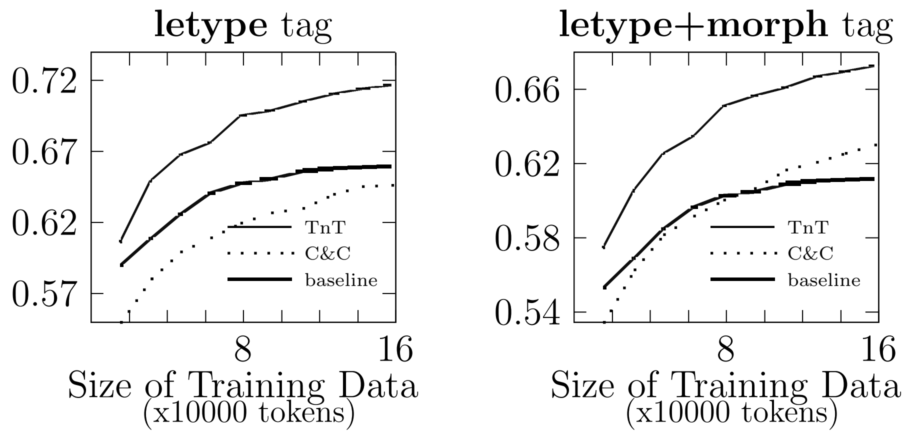


(c) Tag accuracy over the *cb* data set

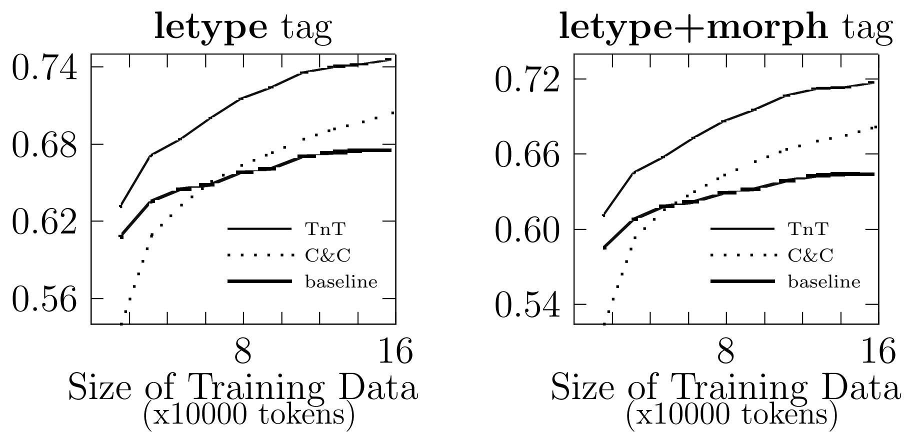
Figure 4.5: Learning curves over different data sets showing accuracy in predicting **letype+sel** and **letype+sel+morph** tags as a function of the amount of training data



(a) Tag accuracy over the *jhpstg*$_t$ data set

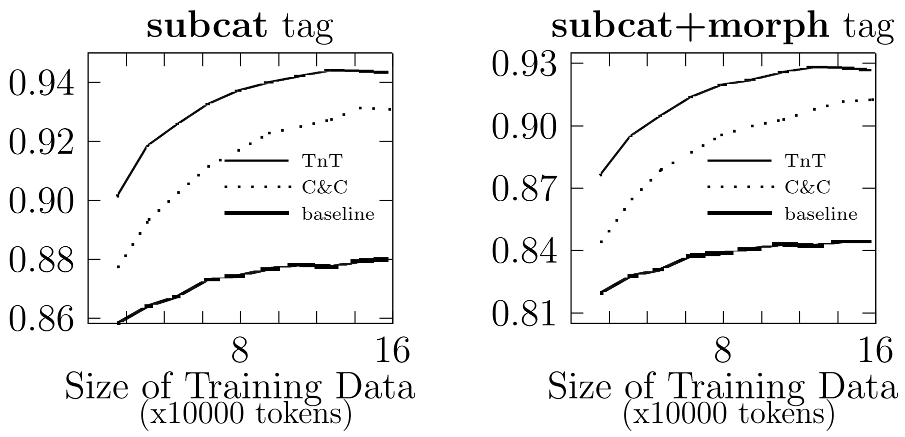


(b) Tag accuracy over the *ws02* data set

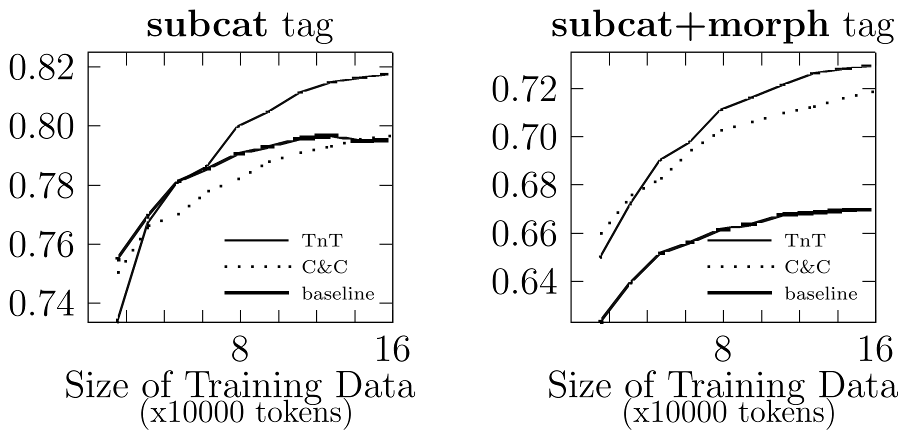


(c) Tag accuracy over the *cb* data set

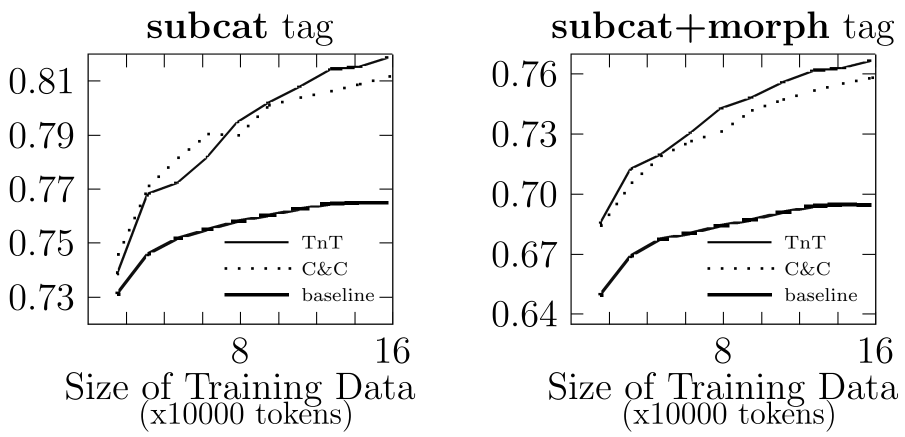
Figure 4.6: Learning curves over different data sets showing accuracy in predicting **letype** and **letype+morph** tags as a function of the amount of training data



(a) Tag accuracy over the *jhpstg*$_t$ data set

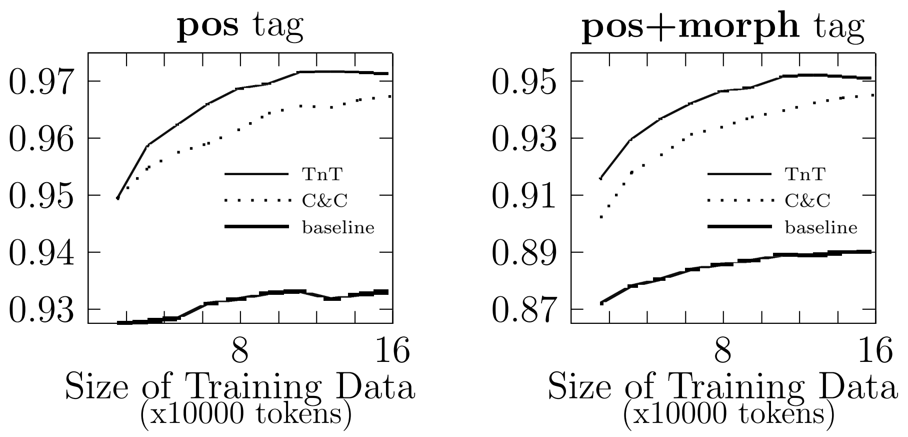


(b) Tag accuracy over the *ws02* data set

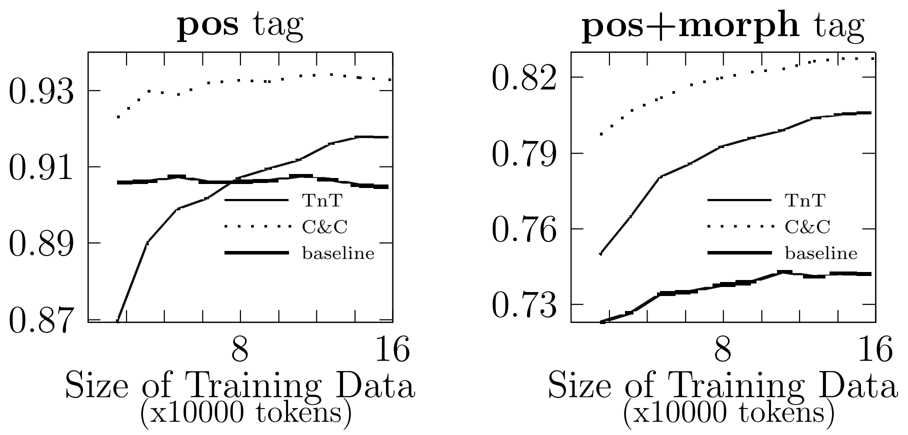


(c) Tag accuracy over the *cb* data set

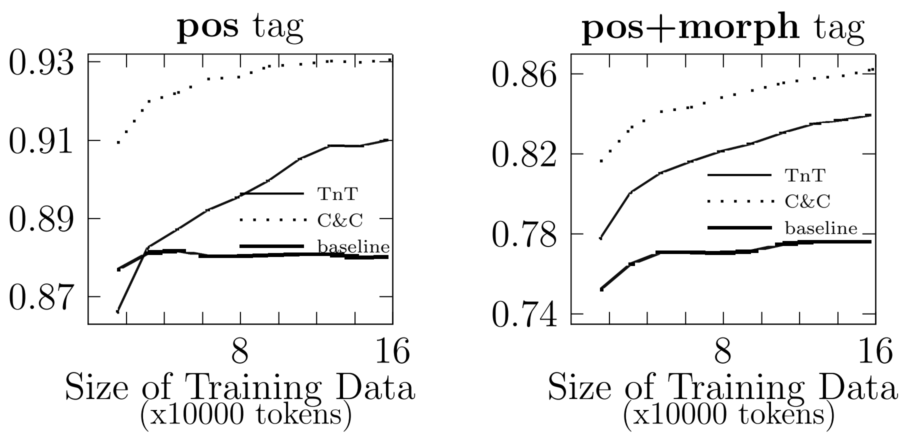
Figure 4.7: Learning curves over different data sets showing accuracy in predicting **subcat** and **subcat+morph** tags as a function of the amount of training data



(a) Tag accuracy over the *jhpstg*$_t$ data set



(b) Tag accuracy over the *ws02* data set



(c) Tag accuracy over the *cb* data set

Figure 4.8: Learning curves over different data sets showing accuracy in predicting **pos** and **pos+morph** tags as a function of the amount of training data

For the in-domain *jhpstg*$_t$ data set, the TnT learning curve has started to flatten out for all tag sets, suggesting we are approaching the upper bound for this style of tagger with approximately 150,000 tokens of in-domain data. For the other two data sets with TnT, and all data sets for the C&C tagger, the learning curves are still rising, indicating that further improvements should be possible given more training data. The baseline tagger, on the other hand, looks to be approaching the limits of its performance, since most of the learning curves for this tagger are flattening.

In order to better understand how the taggers were performing, we looked at the different confusion matrices, focussing on those tags that appeared at least 20 times in the gold data set which might be expected to show stable error patterns. The first observation to be made from this examination is that most of these tags were accurately predicted, meaning, not surprisingly, that rare tags were the ones that were difficult to predict. Within the more frequent tags, there were very few regular confusions — when a tag was incorrectly predicted, the erroneous tag varied almost randomly. A few general trends were, however, observed.

Within the **pos** tag types, *cm* (complementiser) was the most difficult to predict, although not the rarest. It was most often mistaken for a *p* (preposition). In the **pos+morph** set, all taggers occasionally found it difficult to correctly distinguish between past, passive and past participle verbs. Equally, there was often confusion over whether a noun was a mass noun, a count noun or could be used as either. It should be noted that, even in these difficult cases, accuracy was usually over 80%. Very little *qualitative* difference was found between the different taggers, with all showing similar error patterns. There were, however, differences between the test sets. In both the *ws02* and *cb* test sets, there was regular confusion between third singular verbs and plural nouns, and between non-third singular verbs and singular nouns. This confusion was not evident in the *jhpstg*$_t$ test set, which reflects the fact that this set had very few unknown words. The TnT tagger uses suffixes to predict the type of unknown words, which explains this tendency in the case of the TnT tagger. For C&C and the baseline model, both depend heavily on the input POS tags when trying to tag an unknown word. In this case, the POS tags were taken from TnT (using the WSJ model) and so, C&C and the baseline also use, indirectly, the word suffix as a feature.

At the **subcat** and **subcat+morph** levels, the most frequent incorrect predictions involve confusions between verb subcategorisation frames. In the $jhpstg_t$ test set, this is most often very difficult distinctions such as deciding whether an NP complement is optional or obligatory (v_np* versus v_np). In the out-of-domain sets we start to see other subcategorisation frames that were just not seen frequently in the training data, and so the confusions are less predictable.

At the **letype** level, many of the frequent errors were between tags that differ only in the description field. This includes examples like av_-_i-vp_le and av_-_s-vp-pr_le, which designate intersective and scopal adverbs respectively, or p_np_ptcl-of_le and p_np_i-nm-poss_le where the preposition defined is *of*, but in one case semantically empty and in the other marking possession. Other errors also involved word specific tags, such as v_np_poss-le and v_vp_has_le which differentiate between main and auxiliary uses of *has*, as in the utterances *She has flowers.* and *She has sung.* This is exactly the sort of distinction where a tagger could be expected to help, since it can usually be decided by the tag of the following word, and the taggers can predict this distinction with an accuracy between 80% and 90%. Whether this is sufficient accuracy to assist in the parsing process will be explored in later chapters.

Selection relations were infrequent enough that no clear patterns were observed in the **letype+sel** or **letype+sel+morph** tag sets that were not seen in the coarser grained sets.

4.4.2 Multiple Tags

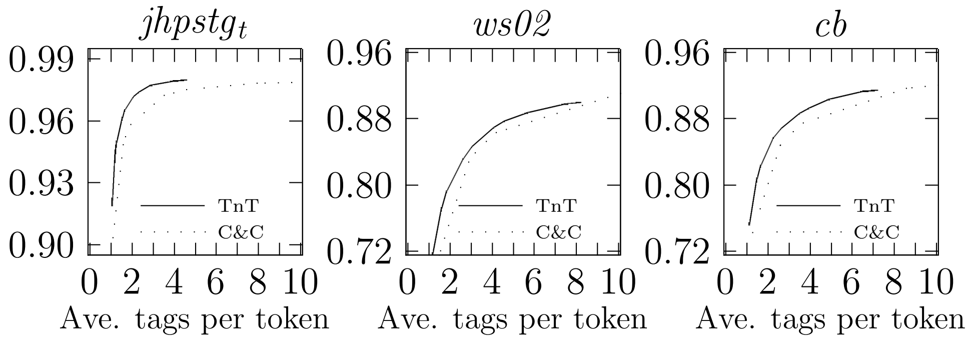
When the output of the supertagger is to be used to restrict the search space, assigning a single tag per token is not necessarily the best option, since it may lead to many unparseable sentences. Clark and Curran (2007b) address this problem by assigning a variable number of tags, dependent on a β value. This β value is used to compare the probabilities of different tags for the same token. When the probability of a tag is with β of the probability of the most likely tag, that tag is also assigned. Both the TnT and C&C taggers have an option to set this β value. In the following experiments, β was varied between 0.5 and 0.00001 (for TnT only), and a token was considered correct if any one of the tags thus produced was the same as the gold tag.

The value of β is relevant only to the particular model and tagger used, and hence, as an external factor, is meaningless. What β indirectly controls is the number of tags assigned to each word. The results of the experiments then graph accuracy against the average number of tags assigned, allowing a meaningful comparison between the two taggers. As we need the probabilities as well as the tags for this evaluation, we need to use the TnT model that matches the tag type, rather than always using the **letype+sel+morph** model, since the probabilities depend on the tag type used in training. This means that TnT will be less accurate when assigning a lower number of coarse-grained tags than the earlier results would suggest. Since the baseline tagger has no probabilistic model at all, it is not included in these experiments. Figure 4.9 shows how accuracy varies with tags assigned for the fine-grained tags, over each data set. The TnT curve in these graphs is always to the left of that from C&C, showing greater accuracy with less tags assigned, with the gap widest for the $jhpstg_t$ in-domain data set. It is clear in most graphs though that the TnT curve is starting to flatten out, while the C&C curve is still increasing. The β values give a possible reason for this: C&C often produces on average 10 or more tags, for a β value of 0.01, while this value for TnT only gives around 4 tags, meaning that C&C is producing a flatter probability distribution. This is not surprising, since the governing principle of Maximum Entropy is that all tags have equal probability, unless information suggests otherwise. Since we have already hypothesised that there is not enough information in the training data for C&C to build an accurate model, a model where many tags have the same probability is to be expected.

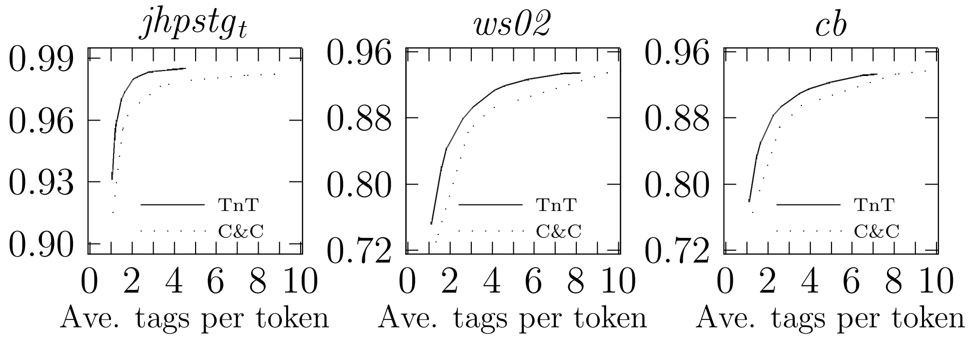
Looking at Figure 4.10 which shows the results for the coarser tags, we see a different picture. Here, as soon as we start to assign more than one tag, C&C is the more accurate tagger. This is particularly clear for the **pos** and **pos+morph** tag sets, where we have already seen that C&C has enough training data to build accurate models. The **subcat** and **subcat+morph** tag sets show that the two taggers give similar results for low numbers of tags, but the gap between the curves widens in C&C's favour once we start to assign more than 2 tags on average.

4.4.3 Selective Tagging

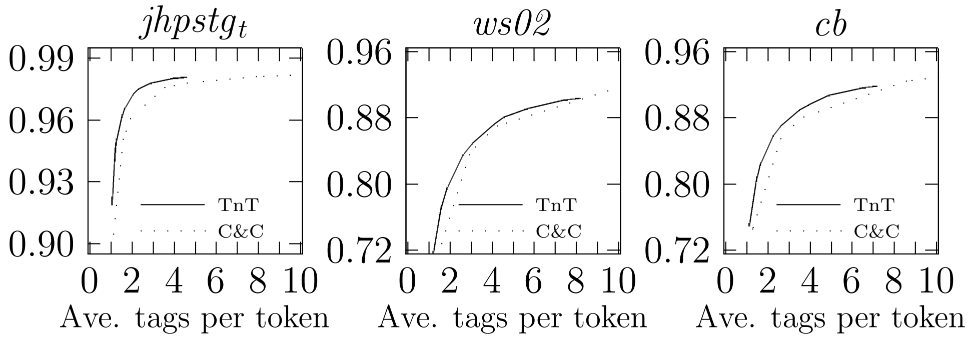
An alternative method of using the supertagger to restrict the search space is by using thresholding to selectively tag. In this scenario, a token is only restricted to a particular supertag when



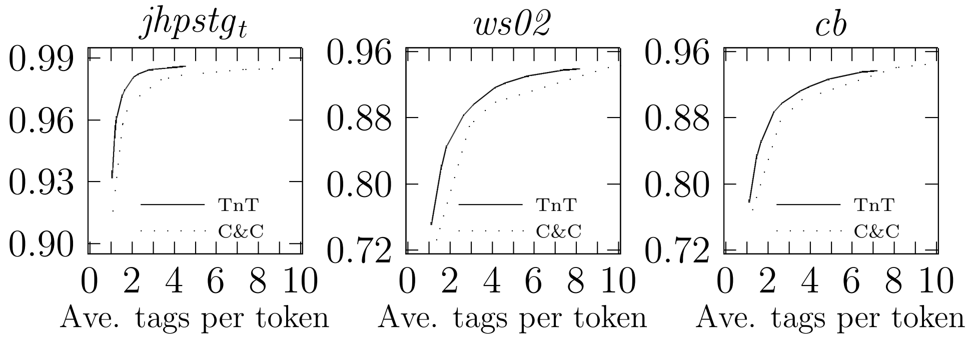
(a) Accuracy over **letype+sel+morph** tags



(b) Accuracy over **letype+sel** tags

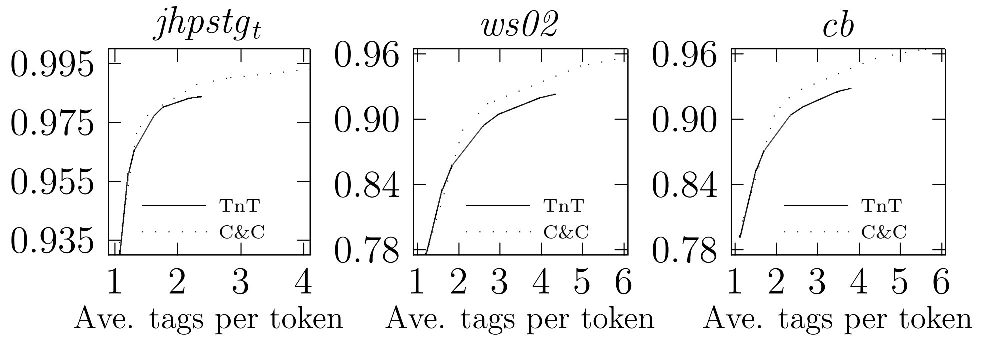


(c) Accuracy over **letype+morph** tags

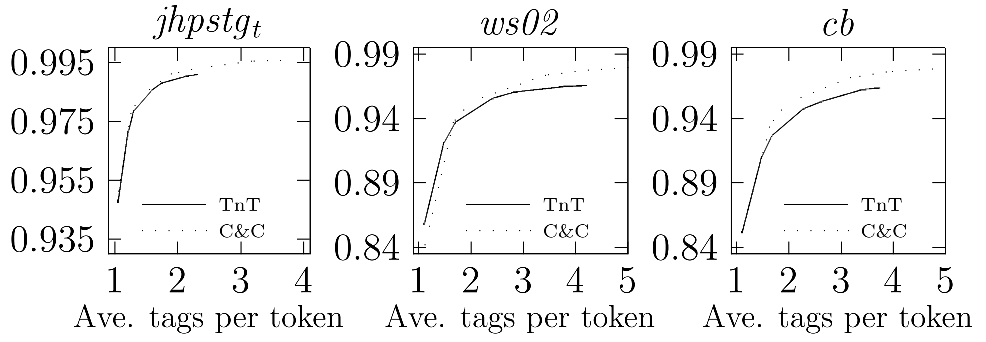


(d) Accuracy over **letype** tags

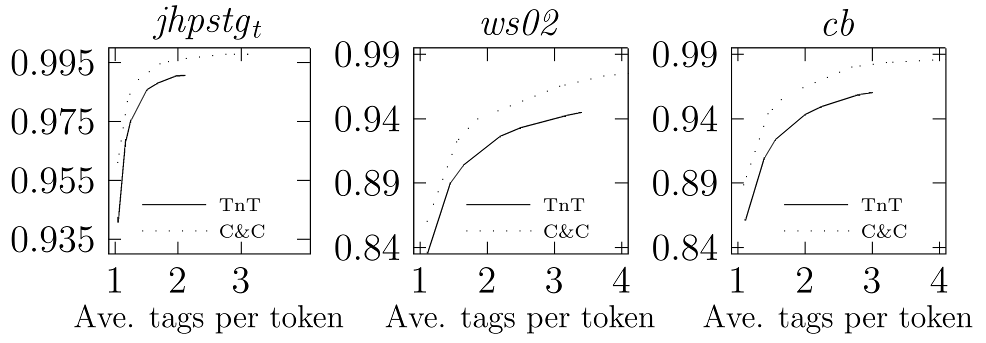
Figure 4.9: Tag accuracy for the fine-grained tags varying against average number of tags assigned (controlled by varying the β value for each tagger).



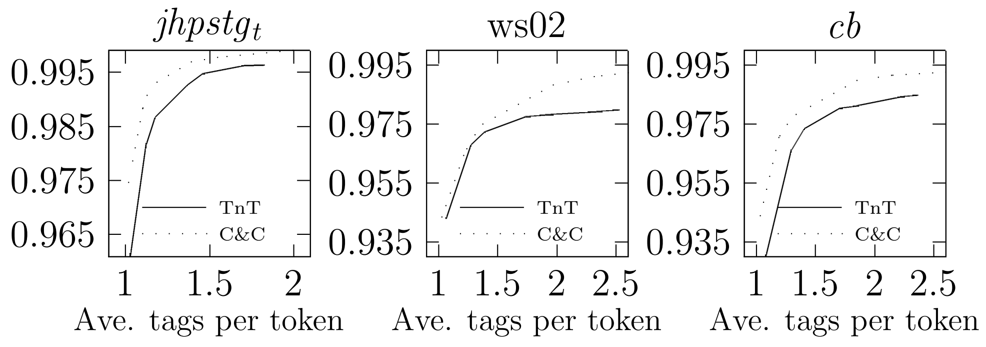
(a) Accuracy over **subcat+morph** tags



(b) Accuracy over **subcat** tags



(c) Accuracy over **pos+morph** tags



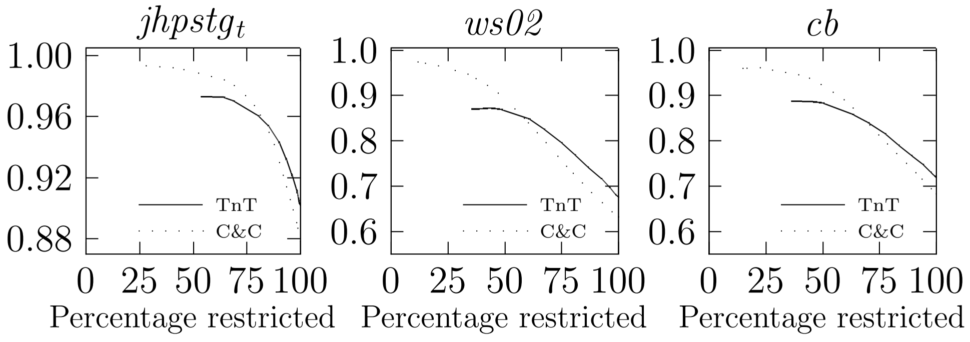
(d) Accuracy over **pos** tags

Figure 4.10: Tag accuracy for the coarse-grained tags varying against average number of tags assigned (controlled by varying the β value for each tagger).

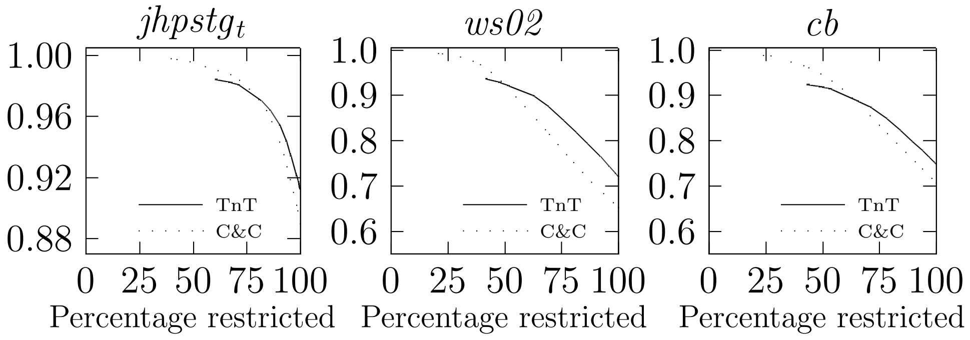
the tag probability is over a certain threshold. This particular method exploits one of the differences between the C&C CCG parser and the PET HPSG parser — in C&C the supertagger and the parser are tightly integrated, while in PET they are separate tools. In C&C, the parser operates directly on the output of the supertagger, since these supertags represent all information known about the word. However, as explained in Section 4.1, an HPSG lexical entry includes word-specific information as well as the category information, and of this, the supertagger can only predict the category information. (Morphological information is added to the lexical entry at a later stage of parsing.) This means that the lexicon must be part of the parsing process and so the supertags are just used as a filter on what comes from the lexicon. As such, it is not necessary for the supertagger to tag every token, since the lexicon already has a filtering effect.

The following experiments evaluate the effect of using different thresholds on both taggers. As in the preceding experiments, the actual threshold value is relevant only to the specific tagger and model. The effect of the threshold is what percentage of tokens will be restricted, and so, in Figures 4.11 and 4.12, results are presented by graphing tag accuracy against percentage tagged (and hence restricted). A high threshold will lead to a low percentage of the tokens being tagged.

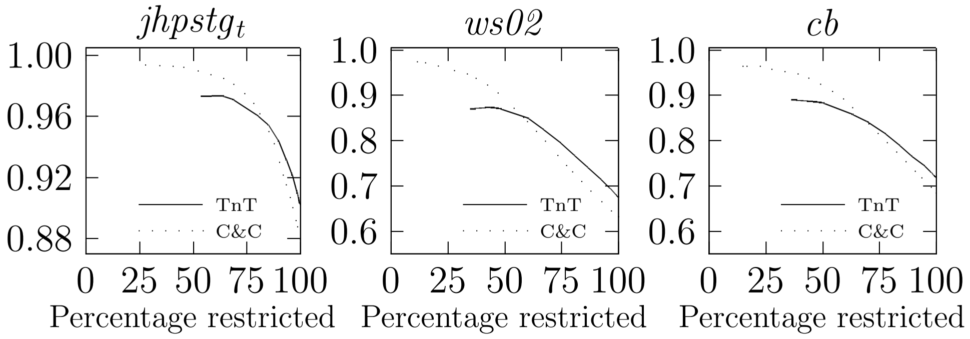
Looking at this representation of the results, we see that C&C is getting very high tag accuracies when it is tagging a low percentage of the tokens. Which tagger is the most appropriate then will come down to the trade-off between efficiency and accuracy — when using fine-grained tags, TnT can more accurately tag a larger percentage of the tokens with fine-grained tags, while C&C can very accurately tag a smaller percentage of tokens. For the coarse-grained tags, again C&C is the more accurate tagger.



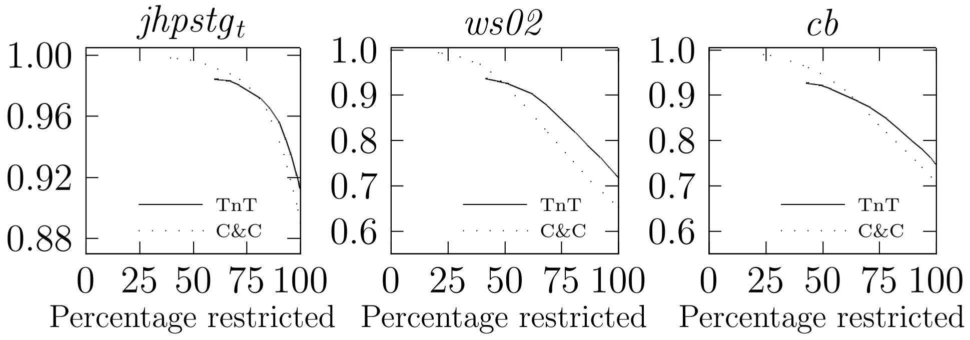
(a) Accuracy over **letype+sel+morph** tags



(b) Accuracy over **letype+sel** tags

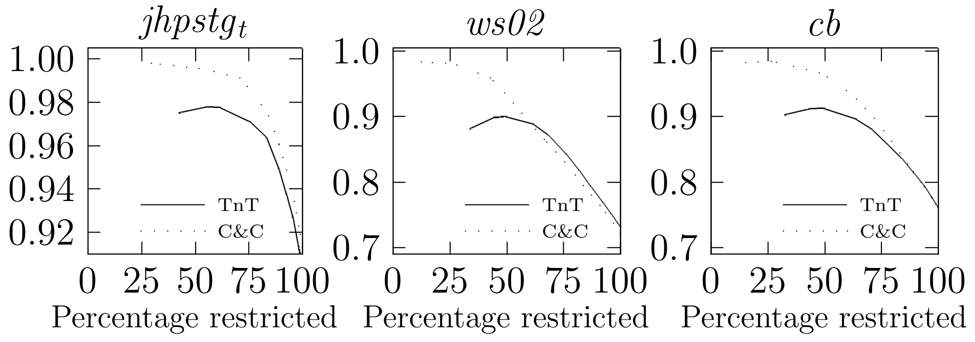


(c) Accuracy over **letype+morph** tags

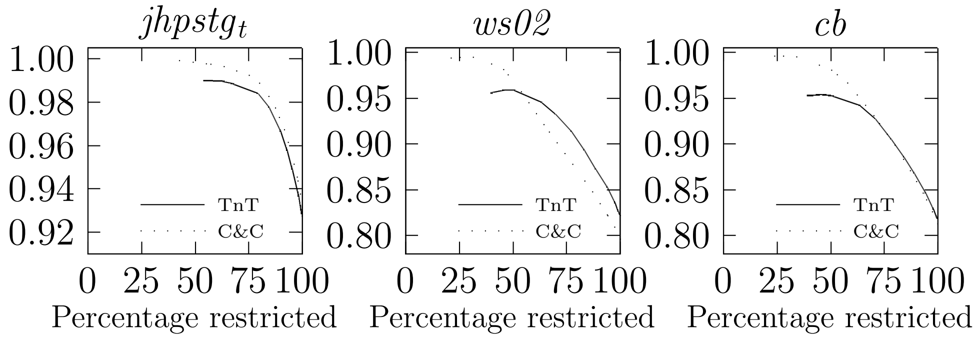


(d) Accuracy over **letype** tags

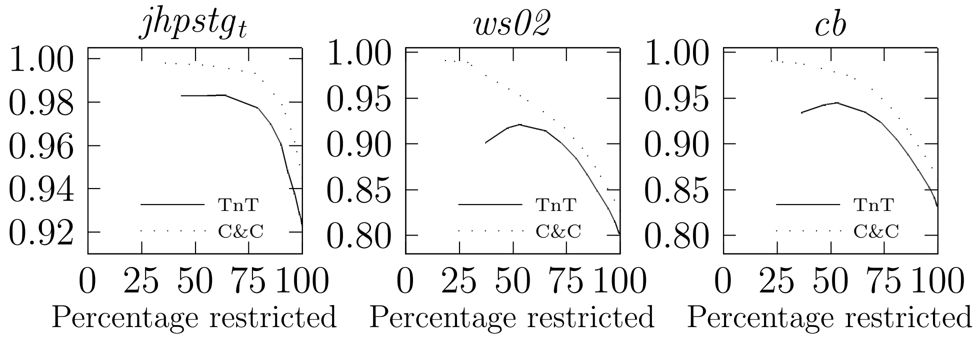
Figure 4.11: Tag accuracy for the fine-grained tags varying against the percentage of tokens restricted (controlled by varying the threshold value for each tagger).



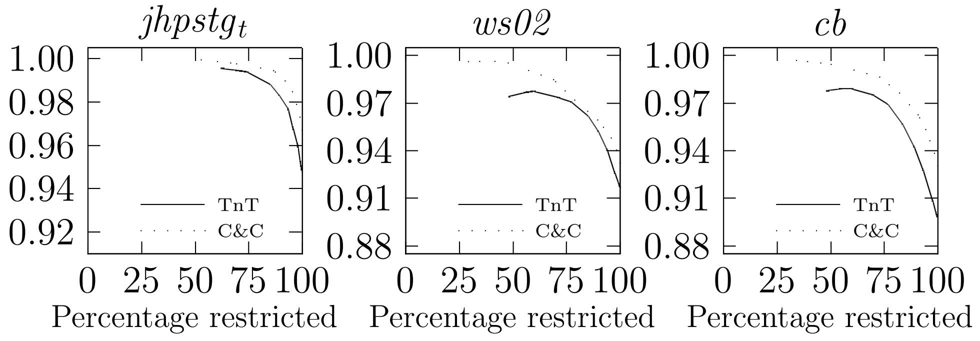
(a) Accuracy over **subcat+morph** tags



(b) Accuracy over **subcat** tags



(c) Accuracy over **pos+morph** tags



(d) Accuracy over **pos** tags

Figure 4.12: Tag accuracy for the coarse-grained tags varying against the percentage of tokens restricted (controlled by varying the threshold value for each tagger).

Whether these coarse-grained tags can restrict the parser search space enough to significantly increase efficiency is something that will be explored in Chapter 6.

4.5 Discussion

The results so far suggest that, on the in-domain test set, all forms of the supertag can be predicted with an accuracy level that has been shown in the past to be sufficient to improve parsing. Clark and Curran (2007b) achieve very good parsing results using a supertagger that has a single tag accuracy of 91.5% and gets around 97.9% when assigning approximately 2 tags per token. The HPSG supertagger used by Matsuzaki *et al.* (2007) in the ENJU parser has a single tag accuracy of 87.5%, which increases to 95.1% when assigning an average of two tags to each token. Both parsers shown significant speed increases when employing these supertaggers. In comparison, the **letype+morph** tag form is the most similar to that used in ENJU, and for this tag type, the TnT tagger has a single tag accuracy of 90.2% and 97.2% when assigning two tags. The most similar tag form to CCG categories is probably **subcat+morph**, where the best single tag accuracy in our experiments is 92.7%, rising to 98.3% when approximately two tags are assigned. Furthermore, the thresholding experiments show another method of search space restriction that could be used when even higher accuracies are required.

Looking at the results on the out-of-domain test set, results are not as encouraging. While domain mis-match is a common issue in all sorts of statistical natural language processing, there are very few published results for out-of-domain supertagging. Rimell and Clark (2008a), as well as Hara *et al.* (2007) describe experiments in adapting their various parsers to alternative domains by retraining their respective supertaggers. Both cases show parser improvement, but only Rimell and Clark gave separate results for the supertagger accuracy. They found that their C&C supertagger, trained on CCGBank had a single tag accuracy of 89.0% for the biomedical domain and 71.6% when tagging questions, compared to the 91.5% the same tagger achieved on Section 0 of CCGBank. Retraining the supertagger model on a small amount (1,328 sentences) of labelled in-domain data brought the tag accuracies up to 93.0% and 92.1% respectively. Question data is notoriously difficult to analyse for NLP tools trained on the more common declarative data, since the syntactic differences are large. On the other hand, domain differences coming from biomedical data, as well as the Wikipedia articles and the technical essay we are using here, are generally attributed more to lexical differences. The tag accuracy differences between in and out of domain data sets in the above experiments are more extreme than those reported by Rimell and Clark, but it is still likely that adding more in-domain data will increase tagger accuracy.

Examining the differences between the two taggers, the clearest suggestion is that TnT can do more with less training data. From the current models, TnT gives the most accurate single tag results for fine-grained tags, where training data was most sparse. On the other hand, C&C is more accurate where sufficient training data is available, such as for the **pos** type tags.

4.6 More Training Data

The overall need so far is for more training data, both to improve out-of-domain performance, and to improve the C&C tagger to a point where the complex model can produce, for fine-grained tags, the sort of accuracy that it achieves on coarse-grained tags. But producing training data is expensive, particularly for the fine-grained tags. The rest of this chapter looks at a few different ways to cheaply boost the training data. The first method emulates that of Rimell and Clark (2008a), by adding a small amount of in-domain data to the larger out-of-domain set. The other methods involve non-gold standard data, obtained either from the taggers themselves, or from the parser.

4.6.1 Domain Adaptation

The results for the *ws02* data set are particularly low, compared to the $jhpstg_t$ set and domain differences probably account for much of that difference. The *ws02* set consists of 11 Wikipedia articles from the domain of Computational Linguistics, very different in topic and style from the LOGON tourism data that has been used for training (Ytrestøl *et al.* 2009). A relatively small amount (1,941 sentences) of parsed and annotated data from the same Wikipedia domain was available to add to the training data. This training data had one problem, however, which was not present in the original data: that of generic lexical entries.

There are two types of generic entry that occur in the treebanks. The first type is used for pattern-based 'named entities' — tokens such as times, dates and email addresses that can be detected by regular expressions. These are detected during the pre-processing stage of parsing so, for example, representations of a time trigger the lexical entry *generic_time_noun_ne* which has a lexical type *n_np_pn-hour-gen_le*. These generic entries are not a problem, since the lexical entry contains all relevant information for use in parsing. The other kind of generic entry is that used for unknown word handling. Chapter 5 will examine this mechanism in more detail, but the basic process is that underspecified lexical entries are triggered by, for example, POS tags. Hence, a POS tag of VB triggers a lexical entry *generic_trans_verb_bse* with a lexical type of *v_np*_bse-unk_le*.

The gold standard for the Wikipedia data, unlike that of the LOGON data, was created using unknown word handling in the parser and 3.3% of the tokens are assigned a generic lexical entry. Given the technical vocabulary contained in these scientific articles, this reasonably high proportion of unknown words is not unexpected. This is increasingly likely to be true, as more treebanks are created from large amounts of real world data, hence any supertagger created from these treebanks will need to address this issue.

Having the supertagger predict these placeholder types, rather than the fully specified lexical types, seems sub-optimal, since ideally we are trying to predict the syntactic properties of a token, not an approximate description, along with the fact that it is unknown to the grammar. The obvious thing to do would be to leave them out of the training data if we do not wish to predict them anyway. However, when there is limited training data available, leaving out every sentence that contained at least one generic type increases the problem. (It is not possible to leave out just the individual tokens, since the taggers predict sequences of tags.) In the case of the Wikipedia data, this would mean leaving out approximately one quarter of the training data. For these experiments therefore, the generic tags were included, unless they could be clearly mapped to an appropriate non-generic type. In some cases this was possible, since there is no meaningful difference between *n_-_pn-unk_le* and *n_-_pn_le*.[1] Unknown verbs, however, were left as *v_np*_bse-unk_le*, even though the subcategorisation information could be wrong, because no further correct information was available.

The taggers were re-trained using the original data and the additional 30,730 tokens of Wikipedia data, bringing the total training data amount to 188,650 tokens. Table 4.7 shows the single tag accuracy over the *ws02* data set for the TnT and C&C taggers with the original model and the new model with the additional Wikipedia data, and also shows the accuracy increase between the two models. Significant increases are shown for all tag types, although still not to the level achieved on the $jhpstg_t$ data set. Using this new augmented model on that $jhpstg_t$ set showed minor increases in accuracy, mostly not statistically significant, but on the *cb* data set, even though it is not clearly the same domain as the new data, significant improvements were seen. Table 4.8 shows these improvements which, while not as impressive as those over *ws02*, are still very good. Whether these improvements are because the Wikipedia data is a good fit for the *cb* data set, or because, at this point, any extra data will yield improvements is not yet clear.

Looking at the differences between the taggers, we still seem to be in the region of the learning curve where TnT is making better use of the training data. TnT, in most cases, shows a greater improvement and has significantly higher accuracy in all except the **pos** and **pos+morph** tag sets where the only statistically significant difference is **pos+morph** over the *ws02* data set. It appears that more data is required to get the full benefit of the C&C tagging model. The next section looks at how that training data might be found.

[1]Details of these mappings are given in Appendix A.

	TnT			C&C		
	original	+Wiki	change	original	+Wiki	change
letype+sel +morph	0.6722	**0.8260**	+15.4%	0.6297	0.7613	+13.2%
letype+sel	0.7163	**0.8631**	+14.7%	0.6458	0.7931	+14.7%
letype +morph	0.6727	**0.8270**	+15.4%	0.6298	0.7623	+13.3%
letype	0.7169	**0.8641**	+14.7%	0.6460	0.7957	+15.0%
subcat +morph	0.7294	**0.8583**	+12.9%	0.7187	0.8187	+10.0%
subcat	0.8177	**0.9124**	+9.5%	0.7967	0.8752	+7.9%
pos +morph	0.8060	**0.8937**	+8.8%	0.8274	0.8808	+5.3%
pos	0.9177	**0.9570**†	+3.9%	0.9328	0.9500†	+1.7%

† marks pairs of results in each row which are not significantly different to each other, statistically

Table 4.7: Single tag accuracy for the *ws02* data set, with the original training set, and then the training set augmented with 30,730 tokens of Wikipedia data

	TnT			C&C		
	original	+Wiki	change	original	+Wiki	change
letype+sel +morph	0.7165	**0.7815**	+6.5%	0.6943	0.7403	+4.6%
letype+sel	0.7453	**0.8075**	+6.2%	0.7024	0.7403	+3.8%
letype +morph	0.7173	**0.7820**	+6.5%	0.6819	0.7417	+6.0%
letype	0.7461	**0.8080**	+6.2%	0.7040	0.7663	+6.2%
subcat +morph	0.7667	**0.8252**	+5.9%	0.7582	0.8029	+4.5%
subcat	0.8188	**0.8647**	+4.6%	0.8117	0.8484	+3.7%
pos +morph	0.8395	0.8825†	+4.3%	0.8623	**0.8832**†	+2.1%
pos	0.9100	0.9323†	+2.2%	0.9306	**0.9381**†	+0.8%

† marks pairs of results in each row which are not significantly different to each other, statistically

Table 4.8: Single tag accuracy for the *cb* data set, with the original training set, and then the training set augmented with 30,730 tokens of Wikipedia data

4.6.2 Using Unlabelled Data

Adding gold standard in-domain data to the training set achieved some significant increases in accuracy, but there is still much need for improvement. Additional gold standard data would be the optimal solution, but gold standard data is expensive and time-consuming to produce. An alternative solution is to use non-gold standard data to train the taggers. In the following sections we explore three different methods to produce this training data. We had two different sources of unlabelled training data available. First, a collection of tourism data from the same domain as the $jhpstg_t$ data set. While TnT appears to be approaching its upper bound for this data set, it is possible that C&C could benefit by adding more in-domain data, even if the data is noisy. There are 23,009 sentences available in this set, making a total of 208,988 tokens. The second source of non-gold standard training data is the ubiquitous Wall Street Journal data from the Penn Treebank. While this data is not in the domain of any of the test sets, it is a large source of data, commonly used in data-driven natural language processing. These experiments will use Sections 3-22,[2] which are comprised of 39,543 sentences, making 939,333 tokens in all.

Self-Training

Self-training has been successfully used as a bootstrapping method for various statistical processing tools such as taggers (e.g. Clark *et al.* 2003) and parsers (e.g. McClosky *et al.* 2006a). In these experiments, we use the tagger models trained on the original data

[2]While Sections 2-21 make up the more traditional training set, some gold standard parsing data was available for Section 2, so this data was held out of the training set. Since there was no variable tuning to require a development set, Section 22 was added to the training data to better approximate the traditional training data amounts.

	TnT		C&C	
Tag Type	Tourism	WSJ	Tourism	WSJ
letype+sel+morph	0.9011	0.7018	0.8777	0.6582
letype+sel	0.9143	0.7411	0.8898	0.6807
letype+morph	0.9015	0.7027	0.8783	0.6578
letype	0.9147	0.7420	0.8908	0.6812
subcat+morph	0.9266	0.7645	0.9125	0.7645
subcat	0.9432	0.8333	0.9309	0.8263
pos+morph	0.9510	0.8357	0.9452	0.8586
pos	0.9713	0.9185	0.9674	0.9466

Table 4.9: Indicative tag accuracies of tagger output for each domain and tag type. The *Tourism* figures are based on tagger accuracies for the $jhpstg_t$ data set, while a small amount (147 sentences) of gold parsed data from WSJ Section 2 was used to estimate tagger accuracy on *WSJ* data.

set to tag our new unlabelled data. This tagged data is then added to the training data, and a new model is trained for each tagger. Of course, the accuracy of the original tagging model will affect how noisy the new training data is. To try and get some indication of that, we looked at how the taggers perform on data from a similar domain. For the new tourism data, the previous results on $jhpstg_t$ should give some indication of the accuracy that could be expected. For the Wall Street Journal (WSJ) data, a small section (147 sentences) of gold standard parsed data from Wall Street Journal Section 2 was available, and we used this for our accuracy estimation. These accuracy indications are shown in Table 4.9.

Tables 4.10, 4.11 and 4.12 show the single tag accuracies on each data set from the two new models, with the accuracy from the original model shown for comparison. For $jhpstg_t$, the effect on TnT performance was mostly negative, particularly for the WSJ data set, though all differences were small. Since the learning curves had suggested that TnT performance was flat-

	TnT			C&C		
	original	+tourism	+wsj	original	+tourism	+wsj
letype+sel	0.9011	**0.9013**	0.8974	0.8777	0.8806	0.8816
+morph letype+sel	0.9143	**0.9160**	0.9122	0.8898	0.8949	0.8945
letype	0.9015	**0.9016**	0.8975	0.8783	0.8809	0.8846
+morph letype	0.9147	**0.9163**	0.9124	0.8908	0.8958	0.8952
subcat	**0.9266**	0.9249	0.9231	0.9125	0.9142	0.9141
+morph subcat	**0.9432**	0.9429	0.9412	0.9309	0.9309	0.9329
pos	**0.9510**	0.9488	0.9470	0.9452	0.9457	0.9452
+morph pos	**0.9713**	0.9696	0.9686	0.9674	0.9678	0.9662

Table 4.10: Single tag accuracy over the *jhpstg*$_t$ data set, with the original model and the two new models trained on the output of the tagger

	TnT			C&C		
	original	+tourism	+wsj	original	+tourism	+wsj
letype+sel	0.6722	0.7340	**0.7890**	0.6297	0.6589	0.7499
+morph letype+sel	0.7163	0.7797	**0.8320**	0.6458	0.6785	0.7837
letype	0.6727	0.7346	**0.7898**	0.6298	0.6606	0.7539
+morph letype	0.7169	0.7803	**0.8327**	0.6460	0.6797	0.7891
subcat	0.7294	0.7814	**0.8262**	0.7187	0.7443	0.8147
+morph subcat	0.8177	0.8620	**0.8932**	0.7967	0.8204	0.8741
pos	0.8060	0.8339	0.8646	0.8274	0.8385	**0.8771**
+morph pos	0.9177	0.9313	0.9434	0.9328	0.9366	**0.9509**

Table 4.11: Single tag accuracy over the *ws02* data set, with the original model and the two new models trained on the output of the tagger

	TnT			C&C		
	original	+tourism	+wsj	original	+tourism	+wsj
letype+sel	0.7165	0.7449	**0.7679**	0.6943	0.6986	0.7380
+morph letype+sel	0.7453	0.7768	**0.7978**	0.7024	0.7193	0.7587
letype	0.7173	0.7454	**0.7683**	0.6819	0.7017	0.7388
+morph letype	0.7461	0.7772	**0.7984**	0.7040	0.7225	0.7624
subcat	0.7667	0.7951	**0.8145**	0.7582	0.7749	0.8024
+morph subcat	0.8188	0.8429	**0.8584**	0.8117	0.8275	0.8488
pos	0.8395	0.8610	0.8758	0.8623	0.8678	**0.8825**
+morph pos	0.9100	0.9249	0.9317	0.9306	0.9334	**0.9385**

Table 4.12: Single tag accuracy over the *cb* data set, with the original model and the two new models trained on the output of the tagger

tening out for this data set, it is not too surprising that we are only adding noise here. For C&C, where lack of training data still appeared to be a problem for the fine-grained tags on this data set, we do get an increase in accuracy for those fine-grained tags. The increases are still quite small though, showing C&C is not getting a lot of benefit from this extra data, even from the in-domain set.

On the *ws02* data set, there are large improvements across the board, but not as large as those from the previous experiment. It appears that 30,730 tokens of gold standard, in-domain data have more effect than even 939,333 tokens of noisy data. It also appears that this extra data is still not enough to lift the performance of C&C over that of TnT for the fine-grained tags.

For the *cb* data set, again we see improvements over the original for all tag types. TnT, with gold standard Wikipedia data, still gets better results, but here we see that the performance of C&C is approximately the same with the 30,730 tokens of gold standard data, or with 939,333 tokens of noisy data. Table 4.9 showed that we might expect less noise if we trained on the output of the TnT tagger, which leads to the idea of co-training between the taggers, explored in the next set of experiments.

Co-Training

Co-training (Blum and Mitchell 1998) is the process of training two learners on the output of each other in an iterative fashion, trying to capitalise on the differences in each learner. Traditional co-training experiments have some way of selecting which output of each tagger is appropriate to use as training for the other, such as the confidence score of the learner. Clark *et al.* (2003) used co-training to boost the performance of the same TnT tagger we use here, together with the C&C Maximum Entropy based POS tagger, which is different to the version of the C&C supertagger we are using, but similar. They experimented with a computationally intensive agreement-based selection method, but found that the naive method of using all the tagged data performed just as well in most cases. We follow their naive co-training method here, as outlined in Figure 4.13, with a cache size of 500 sentences. Clark *et al.* found that co-training was effective when only a small amount of labelled data was available, but achieved no significant improvement when a large (40,000 sentence) labelled seed was used. The seed we use here is much larger than their small seeds (11,175 sentences compared to 50 or 500), but not as large as their large seed. Furthermore, they were predicting POS tags and they could achieve state-of-the-art accuracy with their large seed. We have already seen that more training data can lead to accuracy improvements for most of our tag types and data sets, and so the following experiments investigate whether co-training is effective for a medium-sized seed. The unlabelled

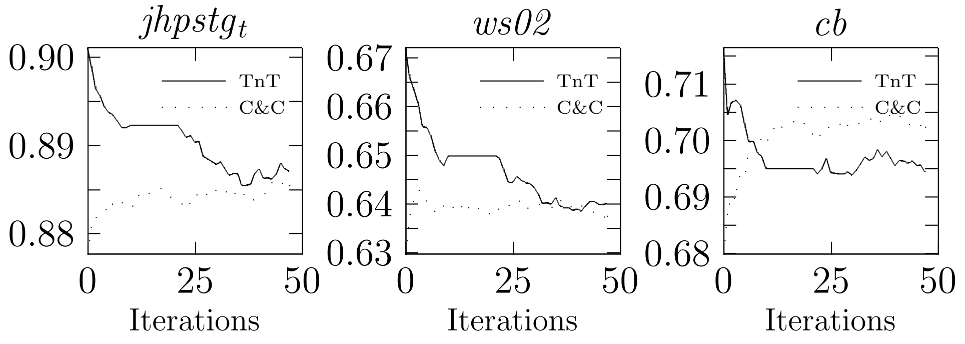
```
$S$ is a seed set of labelled sentences
$L_T$ is labelled training data for TnT
$L_C$ is labelled training data for C&C
$U$ is a large set of unlabelled sentences
$C$ is a cache holding a small subset of $U$

$L_T \leftarrow L_C \leftarrow S$
Train TnT and C&C on $S$
**repeat**
    Partition $U$ into disjoint sets $C$ and $U'$
    $C_T \leftarrow C$ labelled with TnT
    $C_C \leftarrow C$ labelled with C&C
    $L_C \leftarrow L_C \cup C_T$
    Train C&C on $L_C$
    $L_T \leftarrow L_T \cup C_C$
    Train TnT on $L_T$
    $U \leftarrow U'$
**until** $U$ is empty
```

Figure 4.13: The co-training process, taken from Clark *et al.* (2003).

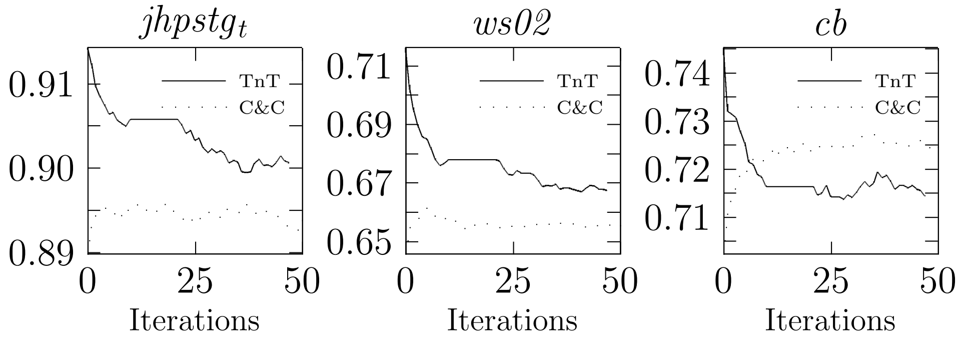
data set is the first 25,000 sentences of the WSJ set used in the self-training experiments.

Figure 4.14 shows that co-training has a degrading effect on TnT accuracy for fine-grained tags. In contrast, the effect on C&C accuracy was negligible or positive. This could be because of their initial accuracy differences, since TnT is training on the less accurate output of the C&C tagger, however the variations in the C&C tagger accuracy don't appear to affect the shape of the TnT curve and Clark *et al.* found that imbalance in initial tagger accuracy did not make co-training less effective.

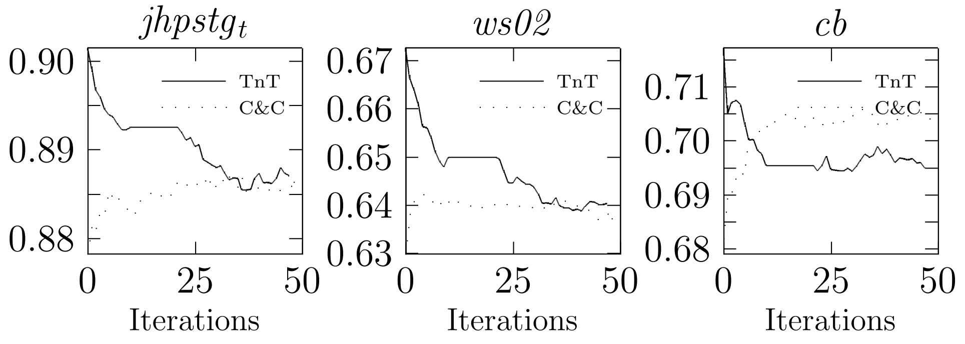
In Figure 4.15, the results for the **subcat+morph** and **subcat** levels of granularity follow this same pattern. Evaluating the **pos+morph** tags, however, C&C performance starts to decrease with co-training, and, for the *cb* data set, co-training has a positive effect on TnT. Again, for the **pos** tags, the accuracy of



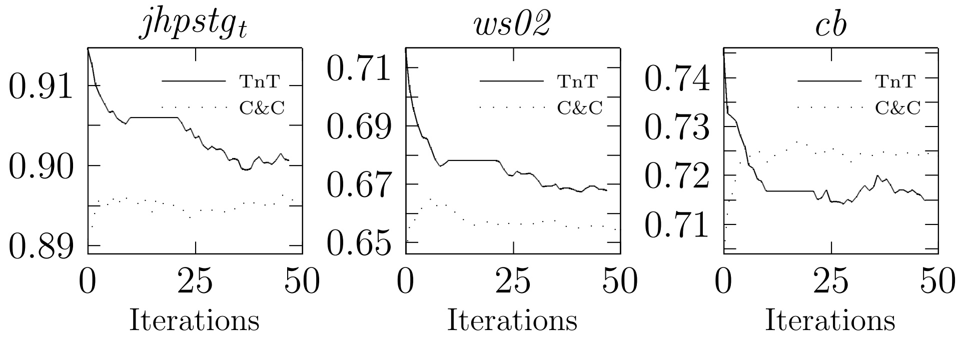
(a) Accuracy over **letype+sel+morph** tags



(b) Accuracy over **letype+sel** tags

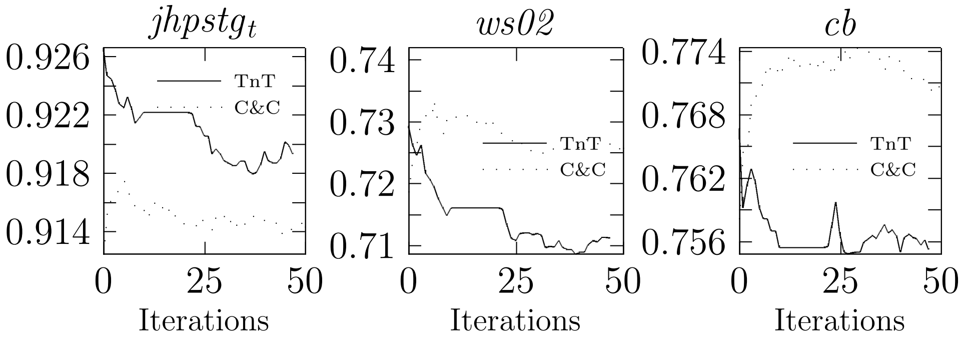


(c) Accuracy over **letype+morph** tags

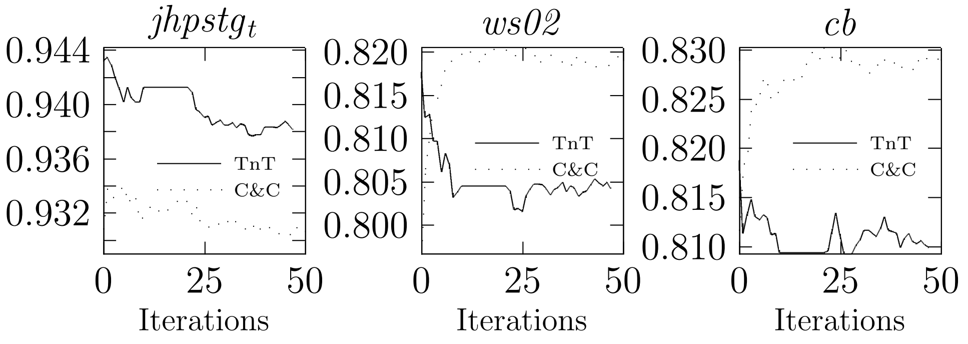


(d) Accuracy over **letype** tags

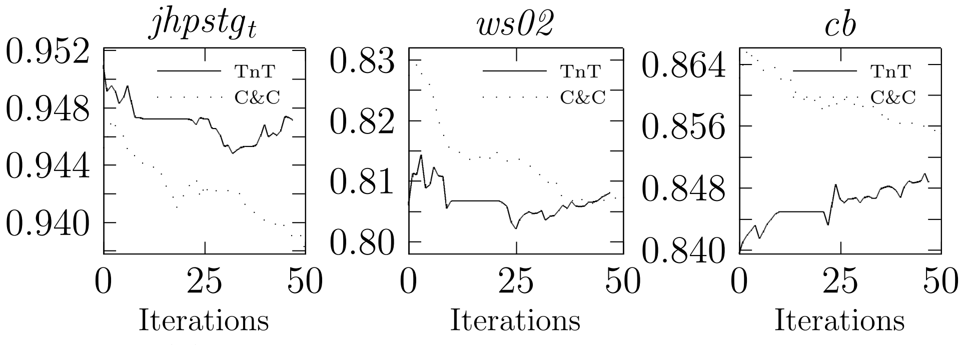
Figure 4.14: Tag accuracy for the fine-grained tags after each iteration of the co-training process.



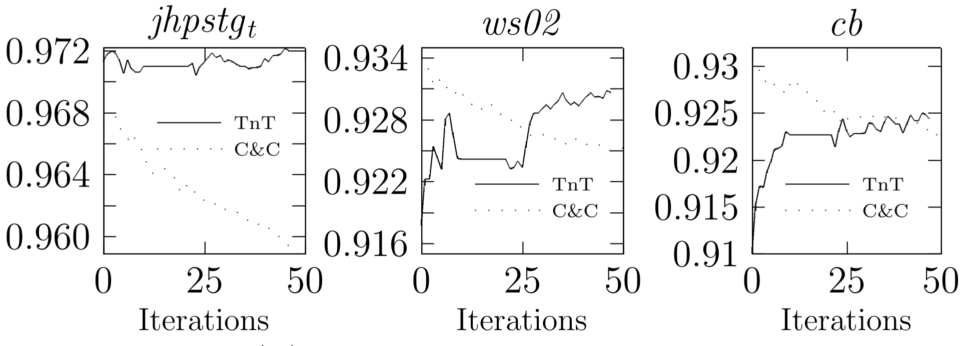
(a) Accuracy over **subcat+morph** tags



(b) Accuracy over **subcat** tags



(c) Accuracy over **pos+morph** tags



(d) Accuracy over **pos** tags

Figure 4.15: Tag accuracy for the coarse-grained tags after each iteration of the co-training process.

the C&C tagger decreases with co-training. In this tag set, TnT accuracy increases for all data sets, but particularly for *ws02* and *cb*, where C&C had the better performance with the initial training data. Our results do suggest that the effect of co-training depends on the accuracy differences of the initial taggers, but we also find that none of the co-training experiments achieve results as good as those from self-training, except for the **pos** tags on the $jhpstg_t$ data set, where none of the differences (between initial, self-training or co-training) are statistically significant. It is possible that a less naive co-training process could get better results by selecting the more accurate labelled instances to train on, but the process here was computationally intensive and adding complications to it would exacerbate that issue. We need another method of producing more accurate labelled data automatically. A candidate is trialled in the next section.

Parser Output

Prins and van Noord (2003) used a different method to produce training data for their supertagger: they used the unannotated output of their parser. Their ALPINO parser, like the PET parser used here, is HPSG-based and it is thought that the strong constraints of the grammar will produce much cleaner data than the taggers alone could. In the following experiments, we parsed the two unlabelled data sets, and used the top parse produced as if it were gold standard data. One disadvantage this method has over self-training or co-training is that not every sentence will be parseable, and hence there will be less additional training data. In this instance, the tourism set provided an extra 19,675 sentences or 175,832 tokens, and the WSJ set an extra 31,253 sentences or 689,028 tokens, 84% and 73% of the potential additional data respectively. The hypothesis is, however, that the

Tag Type	Tourism	WSJ
letype+sel+morph	0.9503	0.9111
letype+sel	0.9593	0.9251
letype+morph	0.9503	0.9111
letype	0.9593	0.9251
subcat+morph	0.9617	0.9293
subcat	0.9713	0.9457
pos+morph	0.9840	0.9527
pos	0.9729	0.9743

Table 4.13: Indicative tag accuracies of parser output for each domain and tag type. The *Tourism* figures are based on the tag accuracy of the top parse for the *jhpstg*$_t$ data set, while a small amount (147 sentences) of gold parsed data from WSJ Section 2 was used to estimate tag accuracy from the parser for the rest of the *WSJ* data.

greater tag accuracy obtainable from parsing will make up for the decrease in the amount of training data. Table 4.13 gives indicative accuracies for the same data sets as Table 4.9. The tag accuracies from the parser should be much higher than those from the taggers, according to these figures. While the figures for the tourism set are highest, since that is the data the parser has been trained on, the accuracy over the WSJ set shows a massive improvement over what the taggers could produce, recommending this as a method for producing training data from out-of-domain labelled data.

Table 4.14 shows the single tag accuracies on the *jhpstg*$_t$ data set. The TnT results here are the highest so far for this data set, with the differences between the model augmented with WSJ data and that using the tourism data statistically insignificant. Likewise, the results from the C&C tagger trained on the WSJ augmented training set, while consistently below those from TnT, are not significantly different.

For the *ws02* data set, Table 4.15 shows that the additional data gave large increases in accuracy, but still not to the level that

	TnT			C&C		
	original	+tourism	+wsj	original	+tourism	+wsj
letype+sel +morph	0.9011	0.9141	**0.9146**	0.8777	0.9045	0.9127
letype+sel	0.9143	0.9270	**0.9283**	0.8898	0.9137	0.9216
letype +morph	0.9015	0.9144	**0.9151**	0.8783	0.9040	0.9126
letype	0.9147	0.9274	**0.9287**	0.8908	0.9143	0.9231
subcat +morph	0.9266	0.9348	**0.9360**	0.9125	0.9267	0.9308
subcat	0.9432	0.9495	**0.9515**	0.9309	0.9431	0.9457
pos +morph	0.9510	**0.9547**	0.9539	0.9452	0.9497	0.9521
pos	0.9713	**0.9719**	0.9717	0.9674	0.9689	0.9689

Table 4.14: Single tag accuracy over the *jhpstg*$_t$ data set, with the original model and the two new models trained on the output of the parser

	TnT			C&C		
	original	+tourism	+wsj	original	+tourism	+wsj
letype+sel +morph	0.6722	0.7275	**0.8149**	0.6297	0.6748	0.7949
letype+sel	0.7163	0.7693	**0.8506**	0.6458	0.6986	0.8300
letype +morph	0.6727	0.7279	**0.8155**	0.6298	0.6791	0.7963
letype	0.7169	0.7696	**0.8512**	0.6460	0.7007	0.8299
subcat +morph	0.7294	0.7785	**0.8457**	0.7187	0.7520	0.8441
subcat	0.8177	0.8537	**0.9021**	0.7967	0.8230	0.8946
pos +morph	0.8060	0.8342	0.8765	0.8274	0.8346	**0.8847**
pos	0.9177	0.9291	0.9402	0.9328	0.9374	**0.9460**

Table 4.15: Single tag accuracy over the *ws02* data set, with the original model and the two new models trained on the output of the parser

	TnT			C&C		
	original	+tourism	+wsj	original	+tourism	+wsj
letype+sel +morph	0.7165	0.7558	**0.8196**	0.6943	0.7226	0.8138
letype+sel	0.7453	0.7864	**0.8430**	0.7024	0.7455	0.8360
letype +morph	0.7173	0.7564	**0.8203**	0.6819	0.7249	0.8149
letype	0.7461	0.7872	**0.8437**	0.7040	0.7481	0.8370
subcat +morph	0.7667	0.8024	**0.8530**	0.7582	0.7837	0.8509
subcat	0.8188	0.8492	0.8863	0.8117	0.8366	**0.8882**
pos +morph	0.8395	0.8669	0.8972	0.8623	0.8704	**0.9022**
pos	0.9100	0.9290	0.9373	0.9306	0.9364	**0.9443**

Table 4.16: Single tag accuracy over the *cb* data set, with the original model and the two new models trained on the output of the parser

the additional gold standard Wikipedia data did. The differences between the taggers for each data set here are not statistically significant for the **subcat+morph**, **subcat**, **pos+morph** and **pos** tag sets, but TnT is still significantly better for the fine-grained tags.

In Table 4.16, we see that the additional parsed WSJ data gives the best results seen so far for the *cb* data set, though these results are still well below the level of accuracy achievable on the $jhpstg_t$ set. The differences between the two taggers for this training set are not statistically significant. Looking at the differences between the two training sets, it is clear that more data is still better, unlike, for example, in the $jhpstg_t$ data set, where both taggers appear to be approaching their upper bound. As results over *ws02* show, in-domain data would be best, but it is not clear which data is in-domain for this *cb* data set. It is online, edited text discussing technical subjects from a social perspective. Not coming from a corpus, it is hard to pin down what text would be similar, but it is exactly the sort of text that one might expect to parse when parsing 'the Web'.

4.7 Summary

Supertags are an effective source of lexical information, in part because they are closely linked to the linguistic formalisms used in parsing. This tight integration with the formalism means that off-the-shelf supertaggers are not readily available for every formalism, and the goal of this chapter was to find an effective supertagger for use with the PET HPSG parser. Since HPSG lexical items are made from information of different types, the definition of supertag form is not straightforward, and eight tag forms of varying granularity were used in the experiments.

The two taggers tested were the HMM-based TnT tagger from Brants (2000b) and the Maximum Entropy based C&C supertagger (Clark and Curran 2007b). Both taggers were trained on data extracted from the gold-standard parse trees released with the HPSG English Resource Grammar, approximately 160,000 tokens. For in-domain tests, this data was sufficient for TnT to reach an tag accuracy level that has been shown to be effective in speeding up parsing. The single tag accuracy of C&C, however, was significantly lower than that of TnT, except when tagging the coarse-grained **pos** and **pos+morph** tag forms. These tags are not strictly speaking supertags, since they provide no more information than Penn Treebank part-of-speech tags, but are included as a baseline. Performance of both taggers dropped considerably when tagging out of domain data, by almost 4% for the **pos** tags, up to over 20% for the most detailed tags. TnT was still the more accurate tagger, for all fine-grained tags.

When multiple tags were assigned, according to a β value, it was possible to get an accuracy of almost 98% on in-domain data, when assigning approximately four of the most detailed tags on average per word. When assigning fine-grained tags, TnT still had the highest accuracy when assigning the same number of tags. With the coarser **subcat** and **subcat+morph** tags, however, the taggers had similar performance when assigning very few tags per word, but C&C was more accurate when assigning more than two tags. Hence we see that the most appropriate tagger depends not only on the tag type, but also on the way the tags are to be used.

Another method of using the tags was evaluated, where a word is only tagged when the tag probability is over a certain threshold. In this scenario, all decisions are left to the parser, unless the tag probability is high. Here we saw that C&C can be the more accurate tagger when not all words need to be tagged, although TnT is generally still the more accurate tagger for fine-grained tags when tagging over 50% of tokens. The optimal tagger and probability threshold depends heavily on the appropriate trade-off point between efficiency (from tagging and hence restricting more tokens) and accuracy (tagging tokens correctly).

Since a lack of training data seemed to be an issue both for out of domain data, and for C&C performance in general, several attempts were made to add extra training data. A small amount of gold standard data in the domain of the *ws02* data set was available, and adding this to the training data gave a large improvement in accuracy for that data set, though not quite to the level achieved on the $jhpstg_t$ set. A significant, though not as large improvement was also seen on the *cb* data set.

Since more gold standard data was unavailable, different methods were tried to automatically produce additional training data. Unlabelled data from the Wall Street Journal, as well as from the tourism domain of the original data, was labelled first by the tagger itself, then as part of a co-training process, labelling training data for one tagger with the other tagger, and finally by parsing with the PET parser. Despite providing less data, since many sentences were not parseable, the data labelled by the parser led to greater tagger accuracy than either self-training or co-training. Table 4.17 shows the best results achieved for each tag type on each data set. For both the $jhpstg_t$ and *cb* data sets, adding large amounts of automatically parsed data to the training data had the best effect on accuracy. The results on *ws02* show that adding gold standard in-domain data is still more effective, since 30,000 tokens of additional Wikipedia data gives better results than almost 700,000 tokens of parsed data.

For a supertagger to be effective, the supertags need to be both predictable, and provide useful information to the parser. This chapter focussed on the first requirement, and found that all tag types are predictable, to varying degrees, for a tagger trained on approximately 150,000 tokens of in-domain data. Later chapters will test the usefulness of different tag forms for different tasks, such as unknown word handling, parser search space restriction and parse ranking. The best method of tagging, the appropriate amount of information in the tag and the required tagging accuracy may all vary according to the task. The following chapter looks at the first of these – unknown word handling.

Data Set	Tag Type	Tagger	Training Data	Accuracy
jhpstg$_t$	**letype+sel+morph**	TnT	+parsed WSJ	0.9146
jhpstg$_t$	**letype+sel**	TnT	+parsed WSJ	0.9283
jhpstg$_t$	**letype+morph**	TnT	+parsed WSJ	0.9151
jhpstg$_t$	**letype**	TnT	+parsed WSJ	0.9287
jhpstg$_t$	**subcat+morph**	TnT	+parsed WSJ	0.9360
jhpstg$_t$	**subcat**	TnT	+parsed WSJ	0.9515
jhpstg$_t$	**pos+morph**	TnT	+parsed tourism	0.9547
jhpstg$_t$	**pos**	TnT	+parsed tourism	0.9719
ws02	**letype+sel+morph**	TnT	+gold Wikipedia	0.8260
ws02	**letype+sel**	TnT	+gold Wikipedia	0.8631
ws02	**letype+morph**	TnT	+gold Wikipedia	0.8270
ws02	**letype**	TnT	+gold Wikipedia	0.8641
ws02	**subcat+morph**	TnT	+gold Wikipedia	0.8583
ws02	**subcat**	TnT	+gold Wikipedia	0.9124
ws02	**pos+morph**	TnT	+gold Wikipedia	0.8937
ws02	**pos**	TnT	+gold Wikipedia	0.9570
cb	**letype+sel+morph**	TnT	+parsed WSJ	0.8196
cb	**letype+sel**	TnT	+parsed WSJ	0.8430
cb	**letype+morph**	TnT	+parsed WSJ	0.8203
cb	**letype**	TnT	+parsed WSJ	0.8437
cb	**subcat+morph**	TnT	+parsed WSJ	0.8530
cb	**subcat**	C&C	+parsed WSJ	0.8882
cb	**pos+morph**	C&C	+parsed WSJ	0.9022
cb	**pos**	C&C	+parsed WSJ	0.9443

Table 4.17: The best single tag accuracy achieved for each data set and tag type.

5 Unknown Word Handling

It is not possible for a grammar to include every word that might be used in a natural language, since, among other reasons, new words are constantly being invented. On the other hand, the strength of lexicalist grammars, like those in the HPSG formalism, is the detailed information about words contained in their lexicons. Robust parsing with lexicalist grammars therefore requires some means of retaining this detailed lexicon while augmenting it to handle those words that the grammar writer has not entered into the lexicon.

In the C&C CCG parser, this task is handled by the supertagger, since, as discussed in Chapter 4, the supertag plus word form completely define the lexicon entries as used by the C&C parser. The DELPH-IN HPSG grammars, with their more fine-grained lexical entries, don't allow this method of augmentation. Instead, the typical mechanism for unknown word handling in DELPH-IN grammars has been to add a number of generic lexical entries to a supplementary lexicon, and allow these generic entries to be triggered by additional information in the input. For the most part, this additional information has been POS tags from a standard POS tagger (e.g., TnT for English and German, ChaSen for Japanese). There is also some previous work (e.g., Baldwin 2005, Zhang and Kordoni 2006, Zhang *et al.* 2007a) that looked at predicting the lexical types of unknown words in DELPH-IN grammars, though most of this work was under the umbrella of deep lexical acquisition and was hence evaluated according to the quality of the augmented lexicons thus produced.

In this chapter, we look at how parsing performance is affected by different forms of unknown word handling when parsing with the ERG. All experiments use the basic mechanism of triggering generic lexical entries for unknown words, with the set of generic entries, and the input information used to trigger them, varying across experiments. The Penn Treebank (PTB) POS tags, as assigned by TnT using a model trained on the Wall Street Journal, have already been shown in recent ERG parsing work to be very effective triggers for this task. They will be used as a baseline, and the major goal of these experiments is to ascertain whether a more accurate analysis can be achieved by using more informative tags or, in the case of our **pos** tags, by using tags more closely aligned to the distinctions made in the grammar.

5.1 Tagger Accuracies Over Unknown Words

In addition to the default PTB POS tags, we will try using some of the different supertag forms from Chapter 4 as triggers for generic lexical entries. Since, in the PET parser, morphological processing is a separate process applied to each lexical entry in the parsing chart after lexicon lookup, we restrict our triggering tags to those without morphological information. Before attempting to parse using these tags, we first look at the standalone accuracy of the different taggers over unknown words. It is common practice to report tagger accuracy on known and unknown words separately, in addition to overall tag accuracy, but that is generally on tokens unknown to the *tagger*. In this case, we are interested in tagger performance on words unknown to the *grammar*.

Evaluating unknown words in this scenario is not straightforward, since the gold standard comes from the parser and hence words unknown to the grammar do not generally appear in the gold standard. As previously mentioned, however, recent gold treebanks were produced by parsing using unknown word han-

Data		**pos**		**subcat**		**letype**		**letype+sel**	
Set	OOV	TnT	C&C	TnT	C&C	TnT	C&C	TnT	C&C
jhpstg$_t$	0.1%	1.00	1.00	0.85	1.00	0.31	0.39	0.31	0.39
ws02	3.1%	0.923	0.931	0.854	0.900	0.490	0.487	0.490	0.490
cb	1.8%	0.909	0.857	0.823	0.806	0.526	0.474	0.526	0.469

Table 5.1: Tagger accuracy for different tag types over those words unknown to the ERG lexicon (OOV). The percentage of tokens unknown in each data set is shown in the second column.

dling and hence some gold trees include generic lexical entries triggered by the default unknown word handling mechanism of using PTB POS tags. While these generic entries are a fuzzy gold standard, they have been accepted in treebanking as producing reasonable analyses. Table 5.1 shows tagger performance over these tags. All experiments use the +WSJ tagger model from Chapter 4, where the base model was augmented with parsed (but not gold standard) data from the Wall Street Journal, except for the TnT experiments over the *ws02* data set. For these experiments we used the +Wiki model, augmented with gold standard Wikipedia data, since this model gave significant increases in overall tagging accuracy in the previous chapter.

In the *jhpstg$_t$* data set, there are only 13 unknown words, with three being mis-spellings, four coming from idiosyncrasies in number pre-processing and three being proper names. Only three words thus are legitimate candidates for addition to the lexicon. The *ws02* and *cb* data sets, on the other hand, both contain a reasonable proportion of unknown words, with nouns making up approximately three quarters of the unknown tokens, and adjectives a further 20%. Unknown verbs and adverbs occur, but infrequently.

Looking at the performance on these tags, we see that the accuracy measured at the **pos** or **subcat** granularity is quite good. For comparison, the TnT POS tagger has an unknown word accuracy of 85.5% tagging Penn Treebank data with Penn Treebank POS tags (Brants 2000a). The performance on the **letype** and **letype+sel** tags, according to these figures, is not as good. However, when examining the 'errors', in many cases the predicted tags are more specific and more accurate than the gold standard tags. For example, many words have been correctly identified as countable nouns, where the gold standard underspecified the tag as a mass or count noun. This underlines the fuzziness of the gold standard, and shows that any evaluation against these gold tags should evaluate for compatibility only, at the **pos** or **subcat** level.

In order to get a better indication of the tagger performance for these fine-grained tags, we need to identify words for which we have gold standard tags, but that resemble unknown words in their properties and distributions. Zhang *et al.* (2007a) describe a fairly complex method of restricting the lexicon according to the distribution of lexical entries in the treebanks and lexemes in a large corpus in order to get controlled levels of correctness. We decided to use the simpler method of assuming any lexical entry not seen in the gold treebanks used for training is unknown by the grammar. In the parsing experiments later in this chapter, we will construct a filtered lexicon based on this assumption. Note that this filtering is based on lexical entries, not words. That is, if *ration* has been seen in the training data as a verb, but not as a noun, we only consider the *ration_n1* lexical entry to be unknown.

According to this definition, just over six percent of the non-punctuation tokens in the gold test sets are now considered 'unknown'. Again, the majority are nouns, making up 57% of unknown tokens, but in this case, 23% are verbs. This discrepancy between the proportion of unknown verbs according to the full

Data		**pos**		**subcat**		**letype**		**letype+sel**	
Set	OOV	TnT	C&C	TnT	C&C	TnT	C&C	TnT	C&C
jhpstg$_t$	5.2%	0.952	0.954	0.914	0.903	0.846	0.801	0.843	0.798
ws02	5.5%	0.820	0.887	0.625	0.782	0.453	0.589	0.448	0.589
cb	8.4%	0.906	0.866	0.782	0.712	0.709	0.611	0.701	0.603

Table 5.2: Tagger accuracy for different tag types over those words unknown to a lexicon that only contains lexical entries that have been seen in the training data (OOV). The percentage of tokens unknown in each data set according to this lexicon is shown in the second column.

lexicon, and according to our filtered lexicon is due, in part, to a recent effort to add verbs to the ERG lexicon. Our filtered lexicon is perhaps representative of a grammar lexicon that has not had this focussed effort on adding verbs. Around 12% of these unknown tokens are adjectives and 6.6% are adverbs. The few closed class ‘unknown’ tokens are mostly from lexicalised multiword expressions such as *on the basis of* (*p_np_i_le*) and *as if* (*p_cp_s-unsp_le*). Since the individual words from these multiword expressions are in the lexicon, removing the multiword lexical entry from the lexicon will not necessarily cause parse failure, but will preclude the precise semantic analysis that the grammar writer has deemed correct for these multiword expressions. Table 5.2 shows the tagger performance over these ‘unknown’ tokens.

Here we see that for *jhpstg*$_t$, performance over the **pos** and **subcat** tags remains high. Moreover, in this instance where we are measuring over real tags rather than those from generic lexical entries, the accuracy over **letype** and **letype+sel** is also very good, particularly from the TnT tagger. In fact, when we look at the differences between the taggers, we see that TnT is significantly more accurate over the *jhpstg*$_t$ and *cb* data sets, in all but one instance, where the difference is statistically insignificant. While we saw, in Chapter 4, that TnT was generally

		Full Lexicon		Filtered Lexicon	
Tagger	Tag Type	+Wiki	+WSJ	+Wiki	+WSJ
TnT	**letype+sel**	0.490	0.481	0.448	0.685
TnT	**letype**	0.490	0.481	0.453	0.686
TnT	**subcat**	0.854	0.883	0.625	0.785
TnT	**pos**	0.923	0.954	0.820	0.875

Table 5.3: Tag accuracy of unknown words in the *ws02* data set for TnT using different training models. The +Wiki model added gold standard Wikipedia data to the basic model and gave the best overall tag accuracy for this data set, for these tagger and tag type combinations. The +WSJ model added parsed (by not gold standard) Wall Street Journal data to the basic model. The +WSJ model was not as accurate overall for this data set, but appears to have better accuracy for unknown words in most cases. The different categories of unknown words are words unknown to the full ERG lexicon and words unknown to a lexicon filtered to include only words seen in the gold treebanks used for training.

the more accurate tagger, the differences had narrowed in most cases to less than one percent when using the augmented models. The much larger differences here indicate that TnT is better at tagging unknown words. Over the *ws02* data set, this trend reverses. One possible explanation for this is that the tagger model TnT used to tag the *ws02* data set was the smaller model trained only on gold data, including gold Wikipedia data. To test this hypothesis, we re-tagged the *ws02* data set with the larger model, augmented with parsed (but not gold standard) Wall Street Journal data. Table 5.3 shows the results. It appears that, even though the model trained on gold standard data does better overall, the bigger, more general model has higher tag accuracy on unknown words. This is an important factor to keep in mind when selecting the best tagger for a particular purpose. All subsequent experiments shall use the model augmented with Wall Street Journal data for both taggers and all data sets.

5.2 Parsing

The next set of experiments uses the tags evaluated above, as well as the standard PTB POS tags as produced by the TnT tagger, to augment parser input for the purpose of unknown word handling. Using the chart mapping mechanism as described by Adolphs *et al.* (2008), the parser accepts Penn Treebank style tokenised input, with up to two associated tags per token, and manipulates the tokens to match the assumptions of the grammar. Some minimal tag assignment adjustment is required in this process. Where multiple input tokens are combined into one, the grammar writer must specify in the chart mapping rules which tags to assign to the combined token. For example, in combining the contracted negations such as {*do*, *n't*} back into one token, the final token is assigned the tag(s) from the first input token, whereas when re-attaching punctuation, the punctuation tags are discarded, regardless of whether they are on the first or last token. An analogous process occurs when single input tokens are split into their component pieces, for example in hyphenated words. In most of these cases, the tags of the original token are assigned to each of the component tokens. Finally, certain tokens are identified, by use of regular expressions, as named entities such as dates, times and URLs, and the tags on these tokens are discarded, since these tokens will trigger another form of lexical entry.

Once the final tagged tokens are settled, separate lexical items are hypothesised for each token and tag combination. These hypothesised lexical items are the ones that may unify with the generic lexical entries that have been specified as part of the grammar. An untagged lexical item for each token is also hypothesised, but this item may only unify with non-generic (referred to as native) lexical entries, and is prevented from unifying with the generic entries. Lexical lookup occurs at this stage of the process, and the final result may be a mixture of lexical items from native lexical entries, and those constructed from generic lexical entries. A lexical filtering process is then applied. In the current ERG, this process throws away all lexical items that come from generic entries if there is one from a native lexical entry that covers the same input span. The only exception to this is if a generic name was hypothesised. These are kept, unless a native lexical entry for the name was included in the lexicon. This filtering process could be more nuanced, retaining some generic items where the original tag had high probability, if the native entry had an incompatible type, but in these experiments, we wish to compare with the status quo. This fact also dictates the manner in which tags are assigned. Chapter 4 showed that the tag accuracy could vary considerably, depending on the number of tags assigned and their relative probabilities. To keep the variation space manageable, we use the tagger configuration currently used to produce the ERG gold treebanks: β is set to 0.01, and furthermore, a maximum of two tags are assigned.

5.2.1 Generic Lexical Entries

The effect of any unknown word handling mechanism is controlled, not only by the triggering tags, but by the form of the generic entries they trigger. A single, extremely underspecified generic entry of type *word* would allow an analysis to be constructed in the vast majority of cases, but would also lead to massive ambiguity (which could in turn lead to a loss of robustness due to resource limitations). By using tags to select particular generic entries, we allow some specification without requiring full knowledge of the word. There are different strategies for selecting which information to specify for a particular tag. One possibility is to specify only that information given in the tag, leaving everything else underspecified. For example, given a tag of *v*, we know only that the word is probably a verb, but know nothing about its subcategorisation, and so a very underspecified lexical entry could be used that specified only the head type as *verb*. However, this strategy ignores characteristics of language, and of unknown words in particular. Another strategy, and the one used in the current grammar, is to define generic lexical entries that specify the most likely information, given that the word is unknown to the grammar. For instance, English has a restricted number of ditransitive verbs, or verbs that take a sentential complement, and any reasonably mature grammar will have most of them in its lexicon. Although all types of base form verb should be assigned the PTB POS tag VB, most *unknown* verbs will be simple transitive, or perhaps intransitive. Hence, by defining the generic lexical entry triggered by VB as a verbal type that takes an optional noun phrase complement, the grammar loses very little coverage, but does not unnecessarily increase ambiguity. This is the strategy we will continue with when defining our new lexical entries.

Designing the appropriate entries to go with the various supertag tag forms is not totally straightforward. The current set of generic lexical entries associated with PTB POS tags have been defined and fine-tuned by the grammar writer over years of grammar development. We will attempt to follow the same basic strategy, but without the same time for fine-tuning.

For the **letype** and **letype+sel** tags, defining the appropriate entries is reasonably simple. Since they are both associated with full lexical types, the lexical entry can inherit from that type and then very little else needs to be specified. For the **letype+sel** tags, the appropriate selectional restrictions are added to the entries. Apart from these details, the only other specification is one intended to make the final lexical items more informative. Since these entries will not, in general, have complete semantics defined for them, they will not have predicate names defined. As such, for most of these generic lexical entries, we set the semantic PRED feature to a string based on the word form and the tag. However, this specification has to be selectively applied, since there are particular lexical types that already define this feature. For example, types referring to pronouns, months, seasons or other named entities use abstract PRED values (e.g. **named_rel**), which are useful for downstream applications such as information extraction and machine translation. These steps lead to the creation of 1006 generic entries for **letype+sel** tags, and 800 for **letype** tags.

Only six generic entries were created for the **pos** tags, for the open class tags **aj**, **av**, **n**, **pp**, **p** and **v**. The other tags describe closed class tags like determiners and conjunctions, which are unlikely to be unknown to a grammar, and whose properties would be very difficult to predict if they were unknown.

For the **subcat** tags, again we only define generic lexical entries for open class tags. Even within this set, there were a number of tags that described subcategorisations that were only used for one word, or a small class of words. No generic entries were created for these types. In the end, 48 generic entries were defined, with the majority representing verb subcategorisations. This was the most difficult set of generic entries to create, since there was frequently conflict between overspecifying information not represented by the tag and underspecifying, and hence increasing parser ambiguity.

5.2.2 Coverage and Efficiency

Since robustness is the main goal of unknown word handling, we first evaluate raw coverage numbers over the full test sets, including those items for which we do not have a gold standard analysis. It is possible, and even likely, that for some items for which a parse is produced, none of the parses will be a reasonable analysis. However, recent treebanking work has found that about 90% of parsed items have at least one correct analysis.

Each data set was parsed with both the full grammar, and the grammar with the filtered lexicon described in Section 5.1. Results from Table 3.3 show how the full grammar performs without unknown word handling. To get the equivalent picture for the filtered grammar, each data set was first parsed with no associated tags on the input tokens. Tables 5.4, 5.5 and 5.6 show the raw coverage for both grammars without unknown word handling as well as one round for each of the different tagger and tag form combinations. Coverage here is defined as the percentage of sentences which received an analysis. The number of sentences that failed to parse either due to exceeding the 60 second time limit or because there were still lexical gaps is shown for each experiment.

When parsing with the full grammar, the default method of triggering generic entries using Penn Treebank POS tags from TnT has the highest coverage for all data sets, although all tag forms achieve a similar level of coverage. Close inspection shows that the number of sentences with lexical gaps is not directly related to coverage (although it obviously has some impact). In some cases it can be seen that, although fewer sentences failed instantly, overall coverage was lower. This occurs when an incorrect tag invokes a generic lexical item that cannot lead to a successful parse.

		Coverage (%)		Timed Out		Lexical Gaps	
Tagger	Tag Types	Full	Filtered	Full	Filtered	Full	Filtered
none	none	93.7	80.6	2	0	22	119
TnT	PTB POS	**96.4**	93.1	2	0	0	0
TnT	**letype+sel**	95.1	91.8	3	2	8	8
C&C	**letype+sel**	95.3	92.6	3	1	6	7
TnT	**letype**	95.1	92.0	3	0	8	8
C&C	**letype**	95.3	92.5	3	2	6	7
TnT	**subcat**	95.8	92.5	2	1	2	2
C&C	**subcat**	96.2	93.6	2	1	0	0
TnT	**pos**	95.9	92.8	2	1	2	2
C&C	**pos**	96.2	**93.7**	2	1	0	0

Table 5.4: Coverage over $jhpstg_t$, parsing with both a full and filtered lexicon. The numbers of sentences which failed to parse due to time outs or lexical gaps are also shown.

		Coverage (%)		Timed Out		Lexical Gaps	
Tagger	Tag Types	Full	Filtered	Full	Filtered	Full	Filtered
none	none	56.8	35.8	5	0	357	547
TNT	PTB POS	**90.9**	85.0	22	12	3	3
TnT	**letype+sel**	89.7	82.5	28	14	3	3
C&C	**letype+sel**	89.4	83.8	28	16	6	7
TnT	**letype**	89.9	82.9	26	13	3	3
C&C	**letype**	89.4	83.8	28	16	6	7
TnT	**subcat**	90.6	84.5	27	17	3	5
C&C	**subcat**	90.1	85.0	31	22	3	3
TnT	**pos**	90.7	85.3	26	15	3	5
C&C	**pos**	90.5	**85.7**	28	18	3	3

Table 5.5: Coverage over *ws02*, parsing with both a full and filtered lexicon. The numbers of sentences which failed to parse due to time outs or lexical gaps are also shown.

		Coverage (%)		Timed Out		Lexical Gaps	
Tagger	Tag Types	Full	Filtered	Full	Filtered	Full	Filtered
none	none	57.1	22.9	14	0	260	550
TnT	PTB POS	**85.3**	72.4	25	12	2	4
TnT	**letype+sel**	83.9	70.0	33	14	1	2
C&C	**letype+sel**	84.1	73.1	33	12	2	7
TnT	**letype**	84.1	70.1	30	14	1	2
C&C	**letype**	84.0	73.1	34	12	2	7
TnT	**subcat**	84.7	71.1	31	19	1	2
C&C	**subcat**	85.0	72.8	32	24	1	1
TnT	**pos**	84.9	71.7	29	18	1	2
C&C	**pos**	**85.3**	**73.6**	29	21	1	1

Table 5.6: Coverage over *cb*, parsing with both a full and filtered lexicon. The numbers of sentences which failed to parse due to time outs or lexical gaps are also shown.

Where the filtered grammar has been used for parsing, the PTB POS tags no longer give the highest coverage. This is not so surprising, since the generic entries for these tags have been specifically tuned to cover those gaps likely, given the grammar. By altering the grammar, we negate the advantage given by this tuning and then the C&C tagger has slightly higher performance, although again the differences in coverage between the different taggers and tag forms are small. We see a slight trend to higher coverage from the coarser grained tags, which is unsurprising, since these tags lead to less constrained lexical items. More surprising is that C&C generally outperforms TnT when assigning the same tag form. The standalone tag accuracies from Section 5.1 would have predicted the reverse trend. One possible explanation is that the method of assigning tags is having an effect. The standalone figures above are for single tag accuracy, and we saw in Chapter 4 that C&C was the more accurate tagger when assigning multiple tags, although the advantage there came when assigning more than two tags. We re-ran the tag evalua-

Data		**pos**		**subcat**		**letype**		**letype+sel**	
Set	OOV	TnT	C&C	TnT	C&C	TnT	C&C	TnT	C&C
jhpstg$_t$	5.2%	0.980	0.977	0.939	0.948	0.898	0.877	0.893	0.871
ws02	5.5%	0.948	0.981	0.871	0.921	0.782	0.727	0.780	0.728
cb	8.4%	0.963	0.963	0.866	0.867	0.795	0.752	0.787	0.735

Table 5.7: Tag accuracy for words unknown to the filtered lexicon (OOV), assigning up to two tags, if the probability of the second tag is at least 0.01 times that of the first tag.

tions, allowing one extra tag if the probability of the second tags was within 0.01 of the first — the same conditions in which an extra tag is assigned during parsing. As the results in Table 5.7 show, the differences between the taggers do narrow, and even slant in favour of C&C for the coarse-grained tags, as we saw in Chapter 4. However, TnT is still the more accurate tagger for fine-grained tags. Even looking in terms of sentence accuracy for these tags shows no advantage to C&C. It appears that, in the complicated interactions in the parsing process, the errors C&C makes have less effect either because they are in sentences that would fail to parse anyway, or because the erroneous tag still allows parsing, although no obvious pattern could be detected in error analysis.

Another interesting pattern in these parsing results can be seen when looking at the number of errors due to time limits, which uniformly drops between the full and filtered grammar results. Generally speaking, generic lexical entries should lead to longer times, because the lexical items they create are underspecified compared to those that come from native lexical entries, and hence lead to more parsing ambiguity. Looking at the details, however, it seems that many sentences that timed out when parsing with the full grammar failed without error with the filtered grammar, probably due to impossible tag sequences being predicted that could not parse. The filtered grammar does, however, show slightly more time outs for the less specific of the new tags, which follows from the idea that underspecification leads to more ambiguity and hence more time.

In terms of absolute coverage numbers, the results for the three data sets reflect the size of the unknown word problem for each set. Despite having a similar percentage of unknown words according to the filtered grammar, the test sets *jhpstg*$_t$ and *ws02* have markedly different coverage when no unknown word handling is used, indicating that the unknown words in the *jhpstg*$_t$ set are clustered in a small number of sentences. For the *cb* data set, parsing of over 70% of sentences fails instantly due to unknown words where no unknown word handling is used. Filtering the grammar had a much greater effect on this data set, and so a top coverage of 73.6%, although perhaps disappointing, is not unexpected.

Unknown word handling is not intended to have any effect on parsing efficiency but, as mentioned above, it is possible for generic lexical entries to lead to longer parsing times, due to underspecification. Of course, parsing times can also increase because we are able to parse more, and longer, sentences. Tables 5.8, 5.9 and 5.10 show the time and memory use while parsing with each of the different tag types.

We can see here that the form of unknown word handling in fact has a large effect on parsing efficiency, largely independent of the coverage achieved. It appears that time and memory usage is related to the number of generic lexical entries that have been added to the grammar. Comparing between taggers with the same tag form, we see that when C&C is used, parsing is always slightly slower, even when TnT gives better coverage. This is most likely because, as we saw in Chapter 4, C&C assigns more

		Time (sec.)		Memory (MB)	
Tagger	Tag Types	Full	Filtered	Full	Filtered
none	none	0.97	0.62	82.74	64.96
TnT	PTB POS	1.10	0.90	104.31	94.88
TnT	**letype+sel**	4.23	4.02	1652.03	1644.88
C&C	**letype+sel**	4.38	4.20	1713.78	1709.43
TnT	**letype**	3.60	3.44	1341.71	1338.61
C&C	**letype**	3.72	3.56	1400.48	1392.91
TnT	**subcat**	1.28	1.28	194.64	190.69
C&C	**subcat**	1.33	1.32	200.86	197.21
TnT	**pos**	1.13	1.10	118.98	113.98
C&C	**pos**	1.17	1.12	118.60	113.51

Table 5.8: Efficiency, measured as average time and memory usage per sentence, over the *jhpstg*$_t$ test data set, parsing with both the full grammar and the filtered grammar.

		Time (sec.)		Memory (MB)	
Tagger	Tag Types	Full	Filtered	Full	Filtered
none	none	2.00	0.54	128.45	66.59
TNT	PTB POS	3.59	2.33	206.57	171.62
TnT	**letype+sel**	8.46	7.26	2400.17	2420.18
C&C	**letype+sel**	9.00	7.93	2509.02	2528.21
TnT	**letype**	7.50	6.25	2077.46	2090.95
C&C	**letype**	7.93	6.87	2193.81	2208.49
TnT	**subcat**	4.40	3.39	369.53	344.58
C&C	**subcat**	4.70	3.97	389.23	372.40
TnT	**pos**	4.12	3.12	240.35	215.53
C&C	**pos**	4.23	3.33	238.44	216.80

Table 5.9: Efficiency, measured as average time and memory usage per sentence, over the *ws02* test data set, parsing with both the full grammar and the filtered grammar.

		Time (sec.)		Memory (MB)	
Tagger	Tag Types	Full	Filtered	Full	Filtered
none	none	3.85	1.17	187.47	93.72
TnT	PTB POS	5.32	3.52	270.41	218.63
TnT	**letype+sel**	11.04	9.08	2711.73	2744.01
C&C	**letype+sel**	11.52	9.69	2804.73	2839.03
TnT	**letype**	9.93	7.91	2360.90	2375.54
C&C	**letype**	10.31	8.50	2453.57	2480.95
TnT	**subcat**	6.35	4.76	454.73	410.73
C&C	**subcat**	6.82	5.65	477.09	444.54
TnT	**pos**	6.05	4.39	313.85	266.68
C&C	**pos**	6.22	4.99	313.60	279.08

Table 5.10: Efficiency, measured as average time and memory usage per sentence, over the *cb* test data set, parsing with both the full grammar and the filtered grammar.

tags than TnT at the same β level, and so more lexical items will be hypothesised. However, the major effect is seen between tag types. Parsing using the **letype+sel** tags, with just over 1000 generic lexical entries, uses at least ten times the memory and twice the time as when PTB POS tags are used. Any benefit from using more specified lexical entries is lost in adding so many entries.

The reason that the size of the supplementary lexicon has such a large effect of parser performance, even in cases where there are very few unknown words, is due to details of the parsing process. At present, lexical instantiation with large numbers of generic entries is implemented in quite an inefficient manner, since most current grammars uses fewer than 20. As described earlier, the supplementary lexicon is searched once for every token and tag combination, a strategy which we will refer to as *gap prevention*. While the parser implementation could be improved if large generic entry sets are to become common, a currently available alternative strategy is to only consult the supplementary lexicon

		Coverage (%)		Time (sec.)		Memory (MB)	
Tagger	Tag Types	Full	Filtered	Full	Filtered	Full	Filtered
none	none	93.6	80.0	0.88	0.55	64.41	48.14
TnT	PTB POS	**96.3**	92.6	0.92	0.80	69.21	61.88
TnT	**letype+sel**	95.1	91.4	0.91	0.99	85.47	172.18
C&C	**letype+sel**	95.3	92.3	0.92	1.04	86.05	185.99
TnT	**letype**	95.1	91.5	0.90	0.96	81.96	147.83
C&C	**letype**	95.3	92.3	0.91	1.00	82.49	159.40
TnT	**subcat**	95.7	92.1	0.95	0.96	73.06	75.60
C&C	**subcat**	96.1	93.2	0.98	1.06	74.17	78.11
TnT	**pos**	95.8	92.4	0.94	0.95	72.28	70.14
C&C	**pos**	96.1	**93.4**	0.98	0.97	72.86	70.68

Table 5.11: Coverage and efficiency over the $jhpstg_t$ data set, parsing with both full and filtered grammars, using a gap detection rather than gap prevention strategy.

after the main lexicon had been searched and a native lexical entry not found. However, this would not allow lexical filtering rules that depended on knowing which generic lexical entries were applicable for a token. With the present lexical filtering policy, this would only affect the rule that keeps lexical items from generic entries that relate to names. Tables 5.11, 5.12 and 5.13 show the coverage and efficiency we get using this alternative strategy, which could be thought of as *gap detection*.

Results show that this gap detection strategy of looking up the generic lexical entries only when no native entry exists is much more efficient, showing reductions in time and memory across the board. The reductions are modest when using PTB POS tags, but substantial for the **letype** and **letype+sel** tags. Comparing coverage on the $jhpstg_t$ data set, we see some small decreases, caused by the fact that some generic items representing names were not included in the parsing chart. Note that this problem does not occur for every unknown name, but only for unknown names which are also real words in the lexicon, for example

		Coverage (%)		Time (sec.)		Memory (MB)	
Tagger	Tag Types	Full	Filtered	Full	Filtered	Full	Filtered
none	none	56.6	35.6	1.57	0.37	92.82	41.59
TnT	PTB POS	**90.5**	85.2	3.08	1.99	140.33	114.77
TnT	**letype+sel**	89.9	82.8	3.30	2.45	316.16	423.13
C&C	**letype+sel**	89.6	84.0	3.42	2.77	334.60	469.53
TnT	**letype**	89.9	83.2	3.25	2.34	279.52	358.39
C&C	**letype**	89.7	83.8	3.38	2.64	297.51	394.13
TnT	**subcat**	90.4	84.4	3.41	2.60	161.54	145.15
C&C	**subcat**	90.0	84.8	3.64	3.15	166.74	159.65
TnT	**pos**	**90.5**	85.3	3.35	2.56	153.91	133.84
C&C	**pos**	90.3	**85.6**	3.56	2.77	155.45	137.37

Table 5.12: Coverage and efficiency over the *ws02* data set, parsing with both full and filtered grammars, using a gap detection rather than gap prevention strategy.

		Coverage (%)		Time (sec.)		Memory (MB)	
Tagger	Tag Types	Full	Filtered	Full	Filtered	Full	Filtered
none	none	56.3	22.6	3.47	1.00	152.49	66.91
TnT	PTB POS	**84.8**	72.2	4.63	2.95	202.91	156.32
TnT	**letype+sel**	84.1	70.4	4.93	3.55	357.40	602.79
C&C	**letype+sel**	84.3	73.0	4.99	3.91	374.79	666.05
TnT	**letype**	84.0	70.5	4.85	3.41	322.39	504.39
C&C	**letype**	84.3	73.1	4.93	3.74	339.20	561.96
TnT	**subcat**	84.4	71.5	5.19	3.76	224.83	201.13
C&C	**subcat**	84.7	73.1	5.54	4.64	235.02	228.30
TnT	**pos**	84.5	72.0	5.15	3.74	218.16	181.95
C&C	**pos**	84.5	**73.7**	5.39	4.34	220.84	197.23

Table 5.13: Coverage and efficiency over the *cb* data set, parsing with both full and filtered grammars, using a gap detection rather than gap prevention strategy.

Brooks. Looking at the other data sets, we see that sometimes coverage increases, and sometimes decreases. Here we see the interplay of two effects: the loss of coverage because unknown names are not always identified, but also the increase in coverage because parsing is more efficient. The gap detection strategy for handling unknown words leads to a reduction of parses that fail due to exceeding the time limit. Whether up or down, the differences in coverage between the two methods is consistently small. The next section will investigate whether there is any difference in accuracy between gap prevention and gap detection, and also examine the effects of the different tag forms on accuracy.

5.2.3 Accuracy

The next evaluations focus on the accuracy of the final analyses. Having seen that, for most tagger and tag form combinations, we can achieve similar coverage to that using PTB POS tags, we now see if there is any benefit in precision to using these new tags. As we saw above, the generic lexical entries that are part of the gold standard treebanks don't really allow evaluation of any type more specific than that used to create the gold standards (the PTB POS tags). Hence, for these accuracy evaluations we parse with the filtered grammar. This leads to more lexical gaps that need to be filled by the unknown word handling mechanism, but in these cases, we have the true gold standard to evaluate against. This evaluation is not a true representation of the ERG performance because of the filtering, but might be assumed to show the effect of the different unknown word handling configurations on a slightly less mature grammar.

Accuracy is measured across those parses for which we have a gold standard analysis, using the EDM metrics described in

		EDM			EDM_{NA}		
Tagger	Tag Types	P	R	F	P	R	F
none	none	0.921	0.724	0.811	0.897	0.704	0.789
TnT	PTB POS	**0.914**	0.859	0.886	**0.886**	0.834	0.859
TnT	**letype+sel**	0.912	0.865	0.888	0.884	0.838	0.860
C&C	**letype+sel**	0.913	**0.866**	**0.889**	0.883	0.838	**0.860**
TnT	**letype**	0.912	0.863	0.887	0.884	0.836	0.859
C&C	**letype**	0.912	**0.866**	**0.889**	0.883	0.838	**0.860**
TnT	**subcat**	0.909	0.854	0.880	0.877	0.828	0.852
C&C	**subcat**	0.908	0.863	0.885	0.875	0.836	0.855
TnT	**pos**	0.907	0.857	0.881	0.875	0.831	0.852
C&C	**pos**	0.907	**0.866**	0.886	0.875	**0.840**	0.857

(a) Gap prevention

		EDM			EDM_{NA}		
Tagger	Tag Types	P	R	F	P	R	F
none	none	0.930	0.723	0.814	0.907	0.704	0.793
TnT	PTB POS	**0.921**	0.858	0.889	**0.894**	0.834	0.863
TnT	**letype+sel**	0.920	0.862	0.890	0.893	0.836	0.863
C&C	**letype+sel**	**0.921**	**0.866**	**0.892**	0.892	0.839	**0.865**
TnT	**letype**	0.920	0.863	0.890	0.892	0.837	0.864
C&C	**letype**	0.920	**0.866**	**0.892**	0.892	0.839	**0.865**
TnT	**subcat**	0.917	0.853	0.884	0.885	0.829	0.856
C&C	**subcat**	0.916	0.862	0.888	0.883	0.837	0.860
TnT	**pos**	0.915	0.856	0.885	0.884	0.831	0.857
C&C	**pos**	0.915	0.865	0.889	0.884	**0.840**	0.861

(b) Gap detection

Table 5.14: EDM evaluation of the top parse for the *jhpstg*$_t$ data set, comparing the accuracy when using a gap prevention (a) or gap detection (b) strategy and a filtered lexicon. Accuracy is measured against all EDM triples, and separately against EDM_{NA} triples, which include NAMES and ARGS triple types, but not PROPS.

Section 3.2.4. Tables 5.14, 5.15 and 5.16 show the comparison for each data set when using a gap prevention or gap detection strategy, including both the full EDM evaluation against all triples, and the EDM_{NA} evaluation, which ignores the PROPS type triples.

		EDM			EDM_{NA}		
Tagger	Tag Types	P	R	F	P	R	F
none	none	0.872	0.270	0.413	0.841	0.261	0.399
TNT	PTB POS	**0.868**	**0.786**	**0.825**	**0.829**	**0.758**	**0.792**
TnT	**letype+sel**	0.857	0.757	0.804	0.817	0.724	0.768
C&C	**letype+sel**	0.856	0.771	0.811	0.815	0.736	0.773
TnT	**letype**	0.856	0.761	0.806	0.816	0.728	0.769
C&C	**letype**	0.856	0.770	0.811	0.815	0.736	0.773
TnT	**subcat**	0.854	0.760	0.804	0.807	0.730	0.767
C&C	**subcat**	0.850	0.759	0.802	0.803	0.728	0.764
TnT	**pos**	0.854	0.774	0.812	0.807	0.743	0.774
C&C	**pos**	0.849	0.771	0.808	0.803	0.742	0.771

(a) Gap prevention

		EDM			EDM_{NA}		
Tagger	Tag Types	P	R	F	P	R	F
none	none	0.873	0.268	0.410	0.843	0.259	0.397
TNT	PTB POS	**0.868**	**0.786**	**0.825**	**0.828**	**0.757**	**0.791**
TnT	**letype+sel**	0.857	0.751	0.800	0.816	0.719	0.764
C&C	**letype+sel**	0.857	0.771	0.811	0.815	0.736	0.773
TnT	**letype**	0.857	0.760	0.806	0.816	0.727	0.769
C&C	**letype**	0.856	0.766	0.808	0.814	0.732	0.771
TnT	**subcat**	0.853	0.755	0.801	0.807	0.725	0.763
C&C	**subcat**	0.849	0.752	0.798	0.802	0.722	0.760
TnT	**pos**	0.854	0.774	0.812	0.807	0.743	0.774
C&C	**pos**	0.849	0.767	0.806	0.803	0.738	0.769

(b) Gap detection

Table 5.15: EDM evaluation of the top parse for the *ws02* data set, comparing the accuracy when using a gap prevention (a) or gap detection (b) strategy and a filtered lexicon. Accuracy is measured against all EDM triples, and separately against EDM_{NA} triples, which include NAMES and ARGS triple types, but not PROPS.

		EDM			EDM$_{NA}$		
Tagger	Tag Types	P	R	F	P	R	F
none	none	0.875	0.174	0.290	0.836	0.165	0.276
TnT	PTB POS	0.846	0.679	0.753	0.795	0.642	0.711
TnT	**letype+sel**	0.847	0.677	0.753	0.796	0.635	0.707
C&C	**letype+sel**	0.847	**0.693**	**0.763**	0.798	**0.652**	**0.717**
TnT	**letype**	**0.848**	0.667	0.747	**0.799**	0.625	0.702
C&C	**letype**	0.847	0.690	0.760	0.797	0.649	0.715
TnT	**subcat**	0.839	0.668	0.743	0.783	0.630	0.698
C&C	**subcat**	0.837	0.679	0.750	0.780	0.641	0.704
TnT	**pos**	0.838	0.676	0.748	0.781	0.637	0.702
C&C	**pos**	0.834	0.683	0.751	0.778	0.647	0.706

(a) Gap prevention

		EDM			EDM$_{NA}$		
Tagger	Tag Types	P	R	F	P	R	F
none	none	0.893	0.173	0.289	0.852	0.164	0.275
TnT	PTB POS	0.854	0.679	0.757	0.802	0.643	0.714
TnT	**letype+sel**	**0.857**	0.662	0.747	**0.806**	0.622	0.702
C&C	**letype+sel**	0.853	**0.688**	**0.762**	0.803	**0.648**	**0.717**
TnT	**letype**	0.856	0.664	0.748	**0.806**	0.624	0.703
C&C	**letype**	0.853	**0.688**	**0.762**	0.803	**0.648**	**0.717**
TnT	**subcat**	0.846	0.667	0.746	0.789	0.629	0.700
C&C	**subcat**	0.843	0.679	0.752	0.786	0.641	0.706
TnT	**pos**	0.844	0.677	0.751	0.787	0.638	0.705
C&C	**pos**	0.841	0.684	0.754	0.785	**0.648**	0.710

(b) Gap detection

Table 5.16: EDM evaluation of the top parse for the *cb* data set, comparing the accuracy when using a gap prevention (a) or gap detection (b) strategy and a filtered lexicon. Accuracy is measured against all EDM triples, and separately against EDM$_{NA}$ triples, which include NAMES and ARGS triple types, but not PROPS.

Comparing between the two strategies, we see that, on *jhpstg*$_t$, gap detection is more accurate. All f-score differences between the two are statistically significant ($p < 0.01$) except results using PTB POS tags, or TnT tagging with **letype+sel** tags. On the other two data sets, the better strategy varies, but none of the differences are statistically significant. In truth, any accuracy difference is not a result of the unknown word handling strategy, but of the effect of the lexical filtering rule that the gap prevention strategy allows. The results here suggest that any benefits from the rule that keeps unknown names in the parsing chart when a native entry is available are not large or consistent enough to make up for the loss of efficiency caused by using gap prevention. It is possible that other lexical filtering rules could add sufficient coverage or accuracy, but with the present policy, gap prevention, at least given the current implementation in PET, does not seem the optimal strategy when parsing for an application. Parsing to produce the gold standards, on the other hand, where the top 500 parses are inspected, could benefit from adding extra items to the chart, and in that case, efficiency is of less concern.

When we turn to look at the differences between tag types, we see that, when comparing precision numbers, there is a trend indicating that the fine-grained tags are more precise than the coarse-grained tags. A high precision can sometimes occur in conjunction with low recall, indicating that the parser is only analysing easy sentences, but that is not the pattern we see here.

Looking at f-scores, the PTB POS tags give the best performance on the *ws02* data set, possibly because of the lower unknown tag accuracy we saw on this data set in Section 5.1. Over the *jhpstg*$_t$ and *cb* data sets, we do see a benefit in accuracy when using the new fine-grained tags, with the C&C tagger giving better results. The difference between the taggers comes from

		Gap prevention			Gap detection		
Tagger	Tag Type	P	R	F	P	R	F
TnT	PTB POS	**0.839**	0.572	0.680	**0.836**	0.572	0.679
TnT	**letype+sel**	0.812	0.633	0.712	0.813	0.621	0.704
C&C	**letype+sel**	0.818	**0.643**	**0.720**	0.819	**0.645**	**0.721**
TnT	**letype**	0.810	0.623	0.704	0.811	0.624	0.706
C&C	**letype**	0.818	**0.643**	**0.720**	0.818	**0.645**	**0.721**
TnT	**subcat**	0.799	0.592	0.680	0.802	0.595	0.683
C&C	**subcat**	0.788	0.623	0.696	0.792	0.626	0.700
TnT	**pos**	0.789	0.593	0.677	0.792	0.596	0.680
C&C	**pos**	0.795	0.640	0.709	0.798	0.643	0.712

Table 5.17: EDM$_A$ evaluation of the *jhpstg*$_t$ data set over just those sentences which have words unknown to the filtered grammar, but are completely know to the full grammar (101 sentences). Results are shown for both the gap prevention and gap detection strategies. Since using generic lexical entries means that predicate names will rarely be right for unknown words, we evaluate only the role relations encoded in the EDM ARGS type triples.

the difference in coverage we noted earlier. While the differences seem consistent, they are largely drowned out by results from sentences with no unknown words, or where the gold standard contains items coming from generic entries. In order to focus on the unknown word performance, Tables 5.17, 5.18 and 5.19 show the results over the subset of sentences for each data set which are affected by filtering the grammar. That is, the sentences that we have a true gold standard for, but that, by filtering the grammar's lexicon, we have forced the parser to employ its unknown word handling mechanism to find an analysis. Since, when using generic lexical entries, it is not possible to get the right predicate name for an unknown word, in this evaluation we only look at the role relations encoded in the EDM ARGS type triples, and ignore differences in the NAMES triples.

Over this subset of sentences, none of the differences between

		Gap prevention			Gap detection		
Tagger	Tag Type	P	R	F	P	R	F
TnT	PTB POS	0.792	0.625	0.699	0.794	0.646	0.712
TnT	**letype+sel**	**0.806**	0.590	0.681	0.800	0.591	0.680
C&C	**letype+sel**	0.798	0.629	**0.704**	0.799	**0.650**	**0.717**
TnT	**letype**	0.804	0.596	0.684	**0.802**	0.613	0.695
C&C	**letype**	0.798	0.629	**0.704**	0.797	0.634	0.706
TnT	**subcat**	0.799	0.601	0.686	**0.802**	0.608	0.691
C&C	**subcat**	0.773	0.618	0.687	0.776	0.620	0.689
TnT	**pos**	0.794	0.622	0.698	0.800	0.646	0.715
C&C	**pos**	0.777	**0.632**	0.697	0.781	0.640	0.703

Table 5.18: EDM$_A$ evaluation of the *ws02* data set over just those sentences which have words unknown to the filtered grammar, but are completely know to the full grammar (157 sentences). Results are shown for both the gap prevention and gap detection strategies. Since using generic lexical entries means that predicate names will rarely be right for unknown words, we evaluate only the role relations encoded in the EDM ARGS type triples.

		Gap prevention			Gap detection		
Tagger	Tag Type	P	R	F	P	R	F
TnT	PTB POS	0.728	0.534	0.616	0.735	0.544	0.625
TnT	**letype+sel**	0.760	0.565	0.648	0.767	0.556	0.645
C&C	**letype+sel**	**0.765**	**0.587**	**0.664**	**0.770**	0.587	**0.666**
TnT	**letype**	0.759	0.551	0.639	0.767	0.556	0.645
C&C	**letype**	0.763	0.581	0.660	**0.770**	**0.588**	**0.666**
TnT	**subcat**	0.736	0.538	0.621	0.742	0.545	0.628
C&C	**subcat**	0.738	0.554	0.633	0.744	0.563	0.641
TnT	**pos**	0.734	0.554	0.632	0.740	0.564	0.640
C&C	**pos**	0.732	0.560	0.634	0.739	0.571	0.644

Table 5.19: EDM$_A$ evaluation of the *cb* data set over just those sentences which have words unknown to the filtered grammar, but are completely know to the full grammar (215 sentences). Results are shown for both the gap prevention and gap detection strategies. Since using generic lexical entries means that predicate names will rarely be right for unknown words, we evaluate only the role relations encoded in the EDM ARGS type triples.

the gap prevention and gap detection strategies are statistically significant. Looking between tag forms however, the differences are clearer than before, and not mixed with those occurring because of generic lexical items in the gold standard. Here we can see that tagging with **letype+sel** or **letype** tags using the C&C tagger gives better performance predicting unknown words. On the *cb* data set in particular, the difference between accuracy with C&C's **letype+sel** and any of the **pos**, **subcat** or PTB POS tags is significant at $p < 0.01$. Over $jhpstg_t$, which is the smallest of the subsets, the differences are significant at $p < 0.05$.

5.3 Summary

The standard unknown word handling mechanism for DELPH-IN HPSG grammars relies on generic lexical entries that are triggered by tags associated with the input tokens. The current method uses Penn Treebank POS tags assigned by the TnT POS tagger and has been shown to achieve very good coverage on unseen data. We experimented with various supertag forms to see whether it was possible to get more precise analyses by using either more detailed tags, or tags that better matched the divisions in the grammar. To do this, we needed to simulate unknown words, since we have no true gold standard for words that are really unknown to the grammar, and so lexical entries were removed from the grammar. The lexical items that were removed from the original lexicon were those not seen in the gold treebanks used for training, which might firstly be considered rare words, and hence likely to be left out of a handwritten grammar. Secondly, this ensured that the words hadn't been seen in the taggers' training data, so no unfair advantage was given to the taggers trained on parser output.

Results when parsing with this filtered grammar showed that using the PTB POS tags still gave good coverage, but that using the other tag forms gave just as good, or better coverage. The one exception was over the *ws02* data set, where lower unknown word tag accuracy meant that the finer-grained tags had slightly lower coverage.

An unexpected result was that the current method of handling unknown words, which we refer to as gap prevention, makes parsing efficiency dependent on the size of the supplementary lexicon used for unknown words. This gap prevention strategy allows the grammar writer great flexibility in controlling which lexical items will be in the parse chart, but, with the current sub-optimal implementation, leads to a massive reduction in efficiency when the number of generic lexical entries is high. At present, this flexibility is not highly utilised, and so we lost very little (if any) accuracy or coverage, by switching to a gap detection strategy and losing this flexibility. Using gap detection, the PTB POS tags were still the most efficient means of dealing with unknown words, but the other tags showed reasonable time and memory usage, and and still achieved coverage as good as, or better than, the PTB POS tags.

Since the supertags did not cause a massive drop in coverage, we looked at whether they brought any benefits in precision and found that, where the tagging was fairly accurate (i.e. for the $jhpstg_t$ and *cb* data sets), the fine-grained tags as assigned by C&C did bring some improvements in accuracy. This is best seen in Tables 5.17, 5.18 and 5.19. We can see here that the **pos** and **subcat** tags also gave a very slight increase over the PTB POS tags, but these differences are not statistically significant.

The method by which we filtered the grammar in order to have a gold standard for unknown words is likely to have a stronger effect on the PTB POS tag unknown word handling, since the generic lexical types associated with those tags have been fine-tuned to match the full grammar. As such, the actual performance over unknown words might be slightly higher using these tags than we have seen. The final conclusion, and the theme that will be continued throughout the thesis, is that the best configuration depends on the exact needs of the application. The PTB POS tags, which are the current default unknown word handling mechanism for the ERG, appear to give the best compromise between efficiency, robustness and accuracy. However, our results do show that **letype+sel** and **letype** tags can be predicted with sufficient accuracy to give good coverage when faced with many unknown words in parser input, and furthermore, produce precise analyses of these unknown words. Thus, if precision is the driving force, using these fine-grained tags might be worth the drop in efficiency. Conversely, where time or memory are limited, using gap detection, rather than gap prevention for unknown word handling appears the better option, even if it comes with a drop in coverage or flexibility. The following chapter will look at another way to increase parser efficiency, which will, in turn, require similar trade-offs.

6 Restricting Parser Search Space

The detailed lexicon at the centre of DELPH-IN HPSG grammars is, at the same time, both the strong point and the weak point of the approach. The detailed information in the lexicon enables the precise semantic analyses that can be produced. On the other hand, as we saw in the last chapter, such level of detail can only be provided for a finite number of lexical entries, and so additional measures must be taken to process input not covered by the lexicon. In this chapter, we address a contradictory weakness arising from the lexicon: rather than insufficient lexical entries, sometimes we have too many. Again, this issue can be attributed to the detailed nature of the lexicon. By providing precise distinct analyses, for example, of *think* in *I think well of him* and *I think about him*, we increase parsing ambiguity for any input involving *think*. Balancing between too many and not enough lexical entries is a problem for any grammar that aims at precise analysis. Furthermore, it is not a direct trade-off between robustness and efficiency. As we saw in Chapter 5, solutions such as gap prevention that add robustness through adding lexical entries to the chart can also decrease robustness by increasing ambiguity and hence increasing failures due to resource exhaustion.

In this chapter, we attempt to increase efficiency by restricting the parser search space. This process is generally referred to in the literature as *supertagging*, but since we are using supertags for multiple purposes, in this work, we limit the term supertagging to mean assigning supertags, and speak separately of the different ways in which supertags can be used. We will restrict parser search space by restricting the number of lexical entries that get added to the parsing chart, which we refer to as lexical restriction. Other possible methods of search space restriction use structure or dependency information from other parsers (e.g. Torisawa *et al.* 2000, Sagae *et al.* 2007, Frank *et al.* 2003, inter alios), or rule probabilities learnt statistically as in any PCFG parser, but in this thesis we focus on benefits possible from using lexical information. Restricting the search space of a parser in this way, by definition, has the potential to decrease robustness by removing lexical entries necessary to the analysis. As mentioned before, however, this is not a straightforward trade-off, since it is also possible to increase robustness by increasing efficiency and hence decreasing failures due to resource exhaustion. The experiments in this chapter will explore this trade-off space.

The first section describes how lexical restriction was implemented in the PET parser. Following that, we describe the potential of this efficiency technique by showing the results obtainable by using gold supertags as described in Chapter 4, and then experiment with different methods of using the supertags within the parser.

6.1 Implementation

The implementation of lexical restriction in PET makes use of the chart mapping mechanism of Adolphs *et al.* (2008) briefly described in Section 5.2. In this scenario, input tokens are feature structures, allowing different attributes of the token to be described precisely. This is, for example, how POS tags are input with each token to be used for unknown word handling. We extended the definition of the token feature structure to add a feature where a list of supertags could be described. Since, as detailed in Chapter 4, the information in the supertags comes from different sources, there are different modes of interaction

$$\text{+STAG} \quad {}_{\text{stag}}\begin{bmatrix} \text{+SLETYPE} & \langle \text{“v\_pp\_e\_le”} \rangle \\ \text{+SSEL} & \langle \text{“\_of\_p\_sel”} \rangle \\ \text{+SMORPHS} & \langle \text{“third\_sg\_fin\_verb\_orule”} \rangle \\ \text{+SPRBS} & \langle \text{“0.89745”} \rangle \end{bmatrix}$$

Figure 6.1: Example of an +STAG feature structure for the supertag **v_pp_e_le+_of_p_sel_rel+third_sg_fin_verb_orule**.

within the parser. To facilitate this, the supertag (+STAG) feature is structured, defining values related to the lexical type, the selectional relations and the morphological information of each supertag. A tag probability can also be included. Figure 6.1 shows an example +STAG feature structure for the tag **v_pp_e_le+_of_p_sel_rel+third_sg_fin_verb_orule**.

The effects of the supertag are controlled at different places within the grammar. The +SLETYPE feature is used for **letype** information, and its generalisations **subcat** and **pos**. To use this information, the lexical type definitions within the grammar are modified to include a reference to the tag values with which they are compatible. This modification can be made individually to each lexical type, which allows the grammar writer a fine level of control. If there are lexical types that should never be restricted, they can be defined to be compatible with all tag values. During lexical lookup in the parsing process, appropriate lexical entries (triggered by their STEM value) are unified with the input token feature structure. Since all lexical entries inherit from a lexical type, an incompatibility between the associated lexical type definition and the +SLETYPE value of the input token feature structure will cause unification to fail, and the lexical entry not to be added to the parsing chart.

The selectional relation information is specified at a lower level than the lexical type: in the lexical entry itself. In order to use this information from the +SSEL feature, the lexical entries need to be modified to require compatibility with the tag value. Then, the same unification process as above will be used to potentially filter out lexical entries based on this information. It is not clear whether this additional information brings much benefit in terms of efficiency, since a filter based on inter-word dependencies (Kiefer *et al.* 1999) operates just after lexical lookup to quickly discard lexical entries with unfulfilled selectional relations. Hence, only scenarios that were not caught by this filter would see any benefit. Our preliminary experiments explore the frequency of this scenario.

As discussed in previous chapters, in the PET parser, morphological processing occurs after lexicon lookup. Hence, to make use of this sort of information from the supertag, a different mechanism from that described above is needed. After potential morphological rules have been applied to the lexical entries, a lexical filtering stage, described in Chapter 5, takes place. In Chapter 5 we saw this lexical filtering used to discard generic lexical entries when native entries occurred in the same input span. In this chapter, we add rules to discard lexical items that have had particular morphological rules applied, if those rules conflict with the value of the +SMORPH feature. By this stage of the parsing process, lexical items with inconsistent morphological rules have already been ruled out, so these filtering rules are only required to decide between morphological rules that could be ambiguous. For instance, an unfiltered parsing chart often contains rules for both the passive and the past tense form of a verb. If the correct form could be accurately predicted, ambiguity could be reduced. However, these are exactly the tags that are likely to be confused in supertagging. Whether tagging accuracy is high enough in these particular instances to improve parsing will be seen in the following sections.

As in unknown word handling, the token mapping process adds a few subtleties to the tag assignments. Following the example of the grammar writer in assigning PTB POS tags, we throw away supertags on punctuation, since punctuation is treated as an affix on word-based tokens. We also discard supertags when generic named entries are identified by pattern matching, since these entries have special handling. In rules controlling the merging or splitting of tokens, case by case decisions were made as to the most appropriate tag assignment for each rule.

Adding extra rules, features and information to the grammar always has the potential to slow parsing down, even when the intended effect was to increase efficiency. A balance needs to be struck between the effectiveness of the change, and added complexity it causes. In the following experiments, we attempt to measure the effects and find a good balance point.

6.2 Baseline and Upper Bound

Since the main goal of lexical restriction is to increase parser speed, we will use, as a baseline, the fastest parser configuration we have seen so far that achieves good coverage. That is, the basic PET parser, using unknown word handling with a gap detection (rather than gap prevention) strategy, and PTB POS tags for triggering generic lexical entries (see Chapter 5 for details). To this basic configuration, we will add the lexical restriction mechanisms, based on supertag forms described in Chapter 4. First, in order to see the potential for this lexical restriction technique, we parsed our test data sets (described in Chapter 3), using gold standard tags of each form. Since we only have gold standard tags where we have gold standard analyses, these first indicative

	Average Lexical Entries					
	per sentence			per token		
Tags	*jhpstg*$_t$	*ws02*	*cb*	*jhpstg*$_t$	*ws02*	*cb*
none	77.70	95.44	121.20	5.95	6.74	6.78
letype+sel	17.87	23.33	26.72	1.37	1.65	1.49
letype	18.12	23.58	27.29	1.39	1.66	1.53
subcat	26.57	32.32	38.80	2.03	2.28	2.17
pos	53.62	61.93	80.95	4.10	4.37	4.53

Table 6.1: Remaining lexical ambiguity when restricting the parse search space to only those lexical entries that match the gold standard tag. These numbers show how many lexical entries are fetched, on average, when restricting by the four basic tag types. (Since **+morph** restrictions occur as a filtering step after lexical entries are extracted, they have no extra effect on these numbers.) The unrestricted numbers (top line) are shown for comparison.

experiments parse only that subset of the test data for which gold analyses are available. Table 6.1 shows the effect the restrictions have on the number of lexical entries that are fetched into the parsing chart. Lexical ambiguity figures are only given for the basic tag types (without **+morph**) because the **+morph** restriction happens as a filtering step after lexical entries are fetched. Tables 6.2, 6.3 and 6.4 show the actual coverage, accuracy and efficiency that results when parsing is restricted with supertags of each separate tag form. Coverage is the proportion of sentences which received an analysis, accuracy is given in terms of EDM f-score, described in Section 3.2.4, and efficiency is given both as average time per sentence (in seconds) and average memory usage (in megabytes). In all cases, the unrestricted numbers are shown in the top line for comparison.

The coverage we see using no restriction is slightly below the 100% that might be expected on these data subsets, given that the options used in this instance were similar to those used to produce the gold standard. The difference comes from the de-

Tags	Coverage	EDM F-score	Time (sec.)	Memory (MB)
none	0.999	0.928	0.67	60.46
letype+sel+morph	1.000	0.959	0.08	23.79
letype+sel	1.000	0.959	0.08	22.58
letype+morph	1.000	0.959	0.08	23.73
letype	1.000	0.959	0.08	22.51
subcat+morph	1.000	0.954	0.10	25.95
subcat	1.000	0.954	0.11	24.98
pos+morph	0.999	0.948	0.21	34.25
pos	0.999	0.948	0.22	33.79

Table 6.2: The space for possible improvement when using lexical restriction over the *jhpstg*$_t$ data set. We show the baseline performance (using no restriction), and the results when using gold standard tags of each tag form to restrict the possible lexical items. Coverage is the proportion of sentences that received an analysis, accuracy is given in terms of EDM f-score, as described in Section 3.2.4, and efficiency is measured as average time and memory usage per sentence.

Tags	Coverage	EDM F-score	Time (sec.)	Memory (MB)
none	0.996	0.887	1.07	83.84
letype+sel+morph	0.999	0.932	0.12	30.66
letype+sel	0.999	0.932	0.12	29.98
letype+morph	0.999	0.932	0.12	30.62
letype	0.999	0.932	0.13	29.93
subcat+morph	0.994	0.923	0.16	33.98
subcat	0.996	0.923	0.18	33.73
pos+morph	0.996	0.916	0.30	44.43
pos	0.997	0.915	0.34	46.20

Table 6.3: The space for possible improvement when using lexical restriction over the *ws02* data set. We show the baseline performance (using no restriction), and the results when using gold standard tags of each tag form to restrict the possible lexical items. Coverage is the proportion of sentences that received an analysis, accuracy is given in terms of EDM f-score, as described in Section 3.2.4, and efficiency is measured as average time and memory usage per sentence.

Tags	Coverage	EDM F-score	Time (sec.)	Memory (MB)
none	0.990	0.886	1.76	116.76
letype+sel+morph	1.000	0.947	0.14	34.16
letype+sel	1.000	0.947	0.14	33.15
letype+morph	1.000	0.947	0.14	34.08
letype	1.000	0.947	0.14	33.07
subcat+morph	0.982	0.931	0.20	38.35
subcat	0.990	0.935	0.20	37.79
pos+morph	0.980	0.916	0.46	56.56
pos	0.988	0.920	0.49	57.81

Table 6.4: The space for possible improvement when using lexical restriction over the *cb* data set. We show the baseline performance (using no restriction), and the results when using gold standard tags of each tag form to restrict the possible lexical items. Coverage is the proportion of sentences that received an analysis, accuracy is given in terms of EDM f-score, as described in Section 3.2.4, and efficiency is measured as average time and memory usage per sentence.

cision to use a gap detection strategy for efficiency reasons. As we saw in Chapter 5, this can occasionally lead to parse failure when a word in the lexicon is being used as a proper name, but is not in the lexicon as such. The gap prevention strategy allows the lexical filtering rule that retains generic lexical entries related to names, where such have been predicted by the tagger, or by heuristics applied during token mapping. With gap detection, the presence of a native entry means no generic options are sought. Interestingly, using lexical restriction can have a similar effect to this filter. By discarding the native entries incompatible with the supertag, in some cases we actually see coverage improvement because a more appropriate generic entry was used in their place.

From the tables, it is clear that lexical restriction has a significant effect in terms of efficiency. In the best case (**letype** and more specific tags on the *cb* data set), we see a twelve-fold speed up. It should be noted, however, that this is not the same speed up potential that has been seen in LTAG and CCG. Bangalore and Joshi (1999) showed that using an oracle supertagger gave them a thirty-fold speed up in parsing with their LTAG parser. Clark and Curran (2004) don't provide parsing results without a supertagger, since their supertagger is an integral part of their C&C CCG parser, however the speed when using single gold supertags compared to using a fairly unrestrictive setting of their supertagger (that supplies many tags per word) is almost 200 times as fast. We won't see these sort of efficiency increases, because, as we see in Table 6.1, the lexical ambiguity is not so high to begin with. On average, less than 7 lexical entries are assigned per word without any restriction, in part because the lexicon itself acts as a filter. In contrast, the CCG supertagger was shown to assign almost 22 categories per word on Section 00 of CCGBank when unrestricted.

Despite not having the massive potential for efficiency increase seen elsewhere, lexical restriction is still worth pursuing as a strategy for increasing parsing efficiency when parsing with the ERG. Even the most general tag form (**pos**) could make parsing three times faster, if it could be accurately assigned. Comparing the effects of different tag forms, we see very little difference between any tag form which uses the full lexical type. All have a parser speed of between eight and twelve times that of unrestricted parsing. However, once lexical entries with incompatible lexical types have been filtered out, the additional filtering from selectional relations or morphological rules has little, if any benefit in terms of time. The **+morph** versions use slightly more memory, but not at a level that makes a discernible difference. Table 6.1 shows that adding the **+sel** information does filter slightly more aggressively, but this is not reflected in the time figures, probably because of the dependency filtering mentioned in Section 6.1.

When the **subcat** generalisation of the lexical type is used for restriction, parser speed is around six to eight times faster than for the unrestricted parser. If tags of this form could be predicted with significantly higher accuracy than the more fine-grained tags, they could provide a good balance between efficiency and robustness. We saw in Chapter 4 that they were more predictable, but whether at a sufficiently higher level to make up for the difference in restrictiveness needs to be determined empirically. At this level of granularity, we see that adding morphological information does bring a slight but consistent increase in speed. Again, we see a slightly higher memory usage for the **+morph** variant, which must be attributed to the additional complexity caused by adding the lexical filtering rules.

Restricting by **pos** or **pos+morph** tags leads to using approximately a third of the parsing time of the unrestricted parser. In these experiments, the speed benefit of adding morphological information is clearly shown. In terms of lexical ambiguity, these tags cause the parser to throw away about a third of the lexical entries that the unrestricted parser considers, but still retain about twice as many as when restricting with **subcat**-based tags. While they are certainly not as effective as the more detailed tags, it could be considered that they provide exactly the information that a tagger should: they discard the blatantly wrong lexical entries, and leave the parser to decide on the fine distinctions made at higher levels. We saw earlier that these tags could be predicted with a very high level of accuracy, and so they would provide the conservative option, increasing speed moderately while not greatly affecting coverage or accuracy.

Another observation from these results is the consistent increase in accuracy. By filtering out incorrect lexical entries, the number of analyses the parse ranking stage has to process is much lower. Note however that even when restricted to analyses which contain the correct sequence of lexical types, accuracy is not 100%, clearly showing (as observed by Toutanova *et al.* (2002)) that the tag sequence is not sufficient to unambiguously determine the correct parse.

The discussions so far focus on the *potential* benefits of each tag form. However, in realistic scenarios there are many variables that may affect the result of using these tags. The next section describes the different ways the tags will be assigned, and also what strategy is taken when the supertag introduces lexical gaps. After that, we will study the results of the different methods and strategies.

6.3 Configurations

In Chapter 4, we described a number of ways in which supertags could be used to restrict parsing. Single tagging is the simplest method, but also the most aggressive. In this method, each token is restricted to be compatible with the single top tag, as produced by the tagger. This will have an effect on lexical ambiguity similar to that shown in Table 6.1, bringing close to upper bound speed. However, any incorrect tag in a sentence will prevent the optimal analysis from being possible, and possibly (depending on the exact tag) prevent any analysis from being found. This is a reasonable method when there are large amounts of data to be parsed and speed is preferred over full coverage, or when the tagger is very accurate.

A more flexible tagging method is multi-tagging, where a variable number of tags are assigned, depending on the tag probabilities. In this method, a β value is used to indirectly control the number of tags assigned to each token, assigning all tags that have a probability with a factor of β of the probability of the top tag. This means that when the top tag has a high probability (and β is high enough), only one tag is assigned, but when the top tag is less likely, more tags can be added.

The third tagging method, selective tagging, assigns a tag only when the tag probability is over a certain threshold. This means that, when the top tag probability is low, we allow all appropriate lexical entries from the lexicon. However, when the tag probability of a token is high, we restrict the lexical entries retrieved for that token. The effectiveness of this method will depend on which tokens can be tagged with a high probability. As we have seen, the lexical ambiguity of the unrestricted parser is around six to seven lexical entries per token, however there is some variability in this figure. Certain word forms, such as *take*, *all* and *in* trigger as much as 30 lexical entries, while many others only have one related entry. Obviously, any benefit from lexical restriction comes about when these highly ambiguous tokens are restricted.

In the following section, we will experiment with the three tag assignment methods: single tagging, multiple tagging and selective tagging. In addition, there will be some investigation of the optimal values of β and the probability threshold in multi-tagging and selective tagging respectively. Apart from these variables, there is another point of variation to consider: what to do when the lexical restriction causes lexical gaps.

Ideally, lexical restriction throws away incorrect lexical entries and keeps the one appropriate entry. However, it is possible to throw away all lexical entries hypothesised for a particular token, either because the tagger was wrong, or because the correct lexical entry was not in the lexicon. There are three potential strategies to use in this scenario. The first is to allow parsing to fail. In a situation where speed is the primary concern, we might choose to say that if the supertagger could not 'select' (through tag compatibility) an appropriate lexical entry from the lexicon, stop attempting to parse and go on to the next sentence. Another option is to fall back to the default unknown word handling mechanism: treat the restriction-triggered lexical gap in the same way as other lexical gaps and use the PTB POS tags to trigger appropriate generic lexical entries. It could be considered inconsistent however, to throw away lexical entries that don't match a very fine-grained tag and instead substitute one or more underspecified entries in that place. Thus, another possibility is to use the supertag to trigger a more specified generic entry, as we saw in Chapter 5. There are many nuanced combinations of these strategies possible, including falling back to different granularities of supertag, but in the following experiments we will stick with the three representative forms of gap handling: no tag, PTB POS tag, and supertag.

6.4 Results

All the following experiments use the supertags as assigned by the C&C and TnT taggers that were described in Chapter 4. We saw in Chapter 5 that the best tagger model could depend on the task at hand, with one model that showed the best overall tagging accuracy having a lower tag accuracy on unknown words. In these experiments, we are attempting to tag every word, and so we again use the +WSJ model described in Chapter 4 for all C&C experiments, and all TnT experiments on the $jhpstg_t$ and *cb* data sets. For TnT experiments on the *ws02* data set, we use the +Wiki model that gave significantly higher overall tag accuracy in Chapter 4. The first set of experiments use the simple single tagging method, and varying the three different gap handling strategies.

6.4.1 Single Tagging

Varying two taggers, eight tag types and three gap handling strategies leads to 48 different possible configurations. Initial examination of the results showed that, for the TnT tagger, there was no speed difference between the **letype**, **letype+morph**, **letype+sel** and **letype+sel+morph** tag types. Since the **letype** tag always had marginally better coverage and accuracy, for the TnT tagger we omit results for the more fine-grained tags. That still leaves 39 configurations, and Tables 6.5, 6.6 and 6.7 show the coverage, efficiency and accuracy for each configuration, along with the results achieved when using no lexical restriction. The different gap handling strategies are indicated by 'none', where gaps cause parse failure; 'PTB', where gaps are covered by generic lexical entries triggered by PTB POS tags, as normal; and 'ST', where the gaps are covered by generic lexical entries triggered by the supertag. We report coverage and efficiency over the full data sets, as well as over the subset for which gold analyses are available. Results over the gold subset allow comparison with Tables 6.5, 6.6 and 6.7, which give upper bounds for each data set. Results are given over the full data sets to measure possible gains on those sets. One potential benefit of lexical restriction is to increase the coverage by decreasing the number of timeouts, and hence we want to see the effects on data which could not be parsed correctly with the baseline setup. Accuracy, in terms of EDM f-score, is measured only over the gold subset.

Configuration			Coverage Full	Coverage Gold	Time Full	Time Gold	EDM F-score
Baseline: no restriction			0.963	0.999	0.93	0.67	0.928
C&C	**pos**	PTB	0.881	0.916	0.24	0.22	0.868
TnT	**pos**	PTB	0.876	0.915	0.23	0.21	0.872
TnT	**pos+morph**	PTB	0.875	0.913	0.24	0.22	0.871
C&C	**pos+morph**	PTB	0.872	0.909	0.25	0.22	0.866
C&C	**pos**	ST	0.869	0.905	0.24	0.22	0.862
TnT	**pos**	ST	0.866	0.907	0.24	0.22	0.867
TnT	**pos+morph**	ST	0.865	0.905	0.24	0.22	0.866
C&C	**pos+morph**	ST	0.862	0.899	0.24	0.22	0.861
TnT	**pos**	none	0.837	0.885	0.22	0.20	0.852
TnT	**pos+morph**	none	0.835	0.884	0.23	0.21	0.851
C&C	**pos**	none	0.801	0.853	0.22	0.21	0.833
C&C	**pos+morph**	none	0.799	0.851	0.22	0.21	0.835
C&C	**subcat+morph**	PTB	0.794	0.840	0.12	0.11	0.813
C&C	**subcat+morph**	ST	0.789	0.831	0.13	0.12	0.810
C&C	**subcat**	PTB	0.781	0.827	0.11	0.10	0.798
TnT	**subcat**	PTB	0.776	0.820	0.11	0.10	0.801
C&C	**subcat**	ST	0.776	0.817	0.14	0.13	0.792
TnT	**subcat+morph**	PTB	0.775	0.819	0.11	0.10	0.800
TnT	**subcat**	ST	0.765	0.811	0.11	0.11	0.793
TnT	**subcat+morph**	ST	0.764	0.809	0.12	0.11	0.793
C&C	**letype**	PTB	0.762	0.809	0.08	0.08	0.775
C&C	**letype+sel**	PTB	0.759	0.804	0.09	0.08	0.771
C&C	**letype**	ST	0.753	0.801	0.23	0.21	0.772
C&C	**letype+sel+morph**	PTB	0.751	0.790	0.09	0.08	0.765
C&C	**letype+morph**	PTB	0.748	0.788	0.09	0.08	0.764
C&C	**letype+sel**	ST	0.748	0.796	0.27	0.25	0.767
C&C	**letype+morph**	ST	0.741	0.783	0.24	0.23	0.762
C&C	**letype+sel+morph**	ST	0.740	0.782	0.28	0.26	0.761
TnT	**subcat**	none	0.724	0.775	0.10	0.09	0.767
TnT	**subcat+morph**	none	0.722	0.774	0.10	0.09	0.766
TnT	**letype**	PTB	0.710	0.754	0.08	0.07	0.740
TnT	**letype**	ST	0.697	0.743	0.14	0.12	0.729
C&C	**subcat+morph**	none	0.667	0.719	0.10	0.09	0.747
C&C	**subcat**	none	0.667	0.719	0.10	0.09	0.732
TnT	**letype**	none	0.651	0.700	0.07	0.06	0.691
C&C	**letype**	none	0.608	0.659	0.06	0.06	0.663
C&C	**letype+sel**	none	0.602	0.652	0.06	0.06	0.659
C&C	**letype+sel+morph**	none	0.586	0.633	0.06	0.06	0.645
C&C	**letype+morph**	none	0.585	0.632	0.06	0.06	0.644

Table 6.5: Coverage, efficiency and accuracy over the *jhpstg*$_t$ data set, when using lexical restriction via a single supertag per token.

Configuration			Coverage Full	Coverage Gold	Time Full	Time Gold	EDM F-score
Baseline: no restriction			0.905	0.996	3.09	1.07	0.887
C&C	**pos**	PTB	0.809	0.907	0.91	0.37	0.803
C&C	**pos**	ST	0.803	0.895	0.90	0.40	0.777
C&C	**pos+morph**	PTB	0.802	0.894	0.91	0.37	0.803
C&C	**pos+morph**	ST	0.793	0.879	0.90	0.39	0.776
TnT	**pos**	PTB	0.757	0.815	0.86	0.33	0.785
TnT	**pos+morph**	PTB	0.747	0.842	0.77	0.34	0.781
TnT	**pos**	ST	0.736	0.831	0.83	0.36	0.752
TnT	**pos+morph**	ST	0.728	0.824	0.74	0.37	0.749
C&C	**subcat+morph**	PTB	0.728	0.819	0.41	0.20	0.730
C&C	**subcat**	PTB	0.725	0.821	0.42	0.20	0.726
C&C	**subcat**	ST	0.721	0.815	0.60	0.27	0.698
C&C	**subcat+morph**	ST	0.720	0.806	0.64	0.26	0.702
C&C	**letype+sel**	PTB	0.659	0.753	0.36	0.15	0.662
C&C	**letype+sel+morph**	PTB	0.653	0.748	0.36	0.16	0.662
C&C	**letype**	PTB	0.640	0.732	0.35	0.15	0.640
C&C	**letype+morph**	PTB	0.638	0.732	0.34	0.16	0.647
C&C	**letype+sel**	ST	0.636	0.725	1.33	0.91	0.641
TnT	**subcat**	PTB	0.629	0.725	0.39	0.16	0.672
C&C	**letype+sel+morph**	ST	0.627	0.715	1.39	0.96	0.633
TnT	**subcat+morph**	PTB	0.625	0.721	0.37	0.16	0.669
C&C	**letype**	ST	0.621	0.707	1.11	0.74	0.624
C&C	**letype+morph**	ST	0.615	0.704	1.15	0.78	0.623
TnT	**subcat**	ST	0.608	0.700	0.54	0.30	0.631
TnT	**subcat+morph**	ST	0.604	0.697	0.51	0.30	0.630
TnT	**letype**	PTB	0.545	0.640	0.28	0.12	0.590
TnT	**letype**	ST	0.544	0.643	0.84	0.53	0.587
TnT	**pos**	none	0.427	0.510	0.36	0.19	0.515
TnT	**pos+morph**	none	0.423	0.506	0.26	0.20	0.513
C&C	**pos**	none	0.393	0.476	0.30	0.20	0.499
C&C	**pos+morph**	none	0.378	0.537	0.28	0.20	0.488
TnT	**subcat**	none	0.299	0.370	0.10	0.07	0.338
TnT	**subcat+morph**	none	0.298	0.368	0.10	0.08	0.337
C&C	**subcat**	none	0.242	0.301	0.11	0.09	0.314
C&C	**subcat+morph**	none	0.240	0.298	0.12	0.10	0.316
TnT	**letype**	none	0.236	0.307	0.06	0.05	0.264
C&C	**letype**	none	0.149	0.183	0.07	0.06	0.177
C&C	**letype+sel**	none	0.148	0.181	0.07	0.06	0.176
C&C	**letype+morph**	none	0.145	0.180	0.08	0.06	0.175
C&C	**letype+sel+morph**	none	0.140	0.173	0.07	0.06	0.163

Table 6.6: Coverage, efficiency and accuracy over the *ws02* data set, when using lexical restriction via a single supertag per token.

Configuration			Coverage Full	Coverage Gold	Time Full	Time Gold	EDM F-score
Baseline: no restriction			0.847	0.990	4.63	1.76	0.886
C&C	**pos**	PTB	0.637	0.787	0.92	0.44	0.737
C&C	**pos**	ST	0.629	0.777	1.01	0.48	0.717
C&C	**pos+morph**	PTB	0.628	0.777	0.90	0.43	0.731
C&C	**pos+morph**	ST	0.614	0.759	0.96	0.48	0.706
TnT	**pos**	PTB	0.610	0.759	0.87	0.43	0.736
TnT	**pos+morph**	PTB	0.597	0.753	0.87	0.44	0.733
TnT	**pos**	ST	0.597	0.742	1.01	0.48	0.717
TnT	**pos+morph**	ST	0.585	0.736	1.02	0.50	0.714
C&C	**subcat**	ST	0.455	0.598	0.54	0.35	0.568
C&C	**subcat**	PTB	0.454	0.594	0.37	0.18	0.576
C&C	**subcat+morph**	PTB	0.445	0.586	0.38	0.19	0.575
C&C	**subcat+morph**	ST	0.440	0.577	0.78	0.48	0.556
C&C	**letype+sel**	PTB	0.433	0.565	0.31	0.14	0.563
C&C	**letype+sel**	ST	0.429	0.555	1.09	0.74	0.537
C&C	**letype**	ST	0.425	0.551	0.93	0.62	0.540
C&C	**letype**	PTB	0.424	0.559	0.30	0.14	0.561
C&C	**letype+morph**	ST	0.421	0.539	0.95	0.62	0.524
C&C	**letype+sel+morph**	ST	0.420	0.541	1.13	0.76	0.522
C&C	**letype+sel+morph**	PTB	0.411	0.537	0.32	0.15	0.531
C&C	**letype+morph**	PTB	0.406	0.531	0.32	0.14	0.527
TnT	**subcat**	PTB	0.388	0.499	0.43	0.22	0.515
TnT	**subcat**	ST	0.385	0.495	0.43	0.22	0.504
TnT	**subcat+morph**	PTB	0.382	0.495	0.33	0.16	0.513
TnT	**pos**	none	0.380	0.513	0.47	0.36	0.543
TnT	**subcat+morph**	ST	0.380	0.491	0.44	0.23	0.502
TnT	**pos+morph**	none	0.377	0.511	0.49	0.37	0.542
C&C	**pos**	none	0.350	0.475	0.46	0.27	0.476
TnT	**letype**	ST	0.342	0.457	0.71	0.43	0.468
C&C	**pos+morph**	none	0.339	0.463	0.46	0.29	0.467
TnT	**letype**	PTB	0.339	0.449	0.25	0.11	0.456
TnT	**subcat**	none	0.229	0.318	0.14	0.11	0.327
TnT	**subcat+morph**	none	0.228	0.316	0.14	0.11	0.326
C&C	**subcat**	none	0.195	0.278	0.12	0.09	0.279
TnT	**letype**	none	0.181	0.262	0.10	0.07	0.261
C&C	**subcat+morph**	none	0.179	0.258	0.11	0.09	0.255
C&C	**letype+sel**	none	0.150	0.211	0.08	0.06	0.199
C&C	**letype**	none	0.144	0.207	0.07	0.06	0.198
C&C	**letype+sel+morph**	none	0.144	0.207	0.08	0.06	0.186
C&C	**letype+morph**	none	0.139	0.199	0.07	0.06	0.176

Table 6.7: Coverage, efficiency and accuracy over the *cb* data set, when using lexical restriction via a single supertag per token.

Results are ordered according to the coverage and we can clearly see that the coarse-grained **pos** and **pos+morph** tags achieve the best coverage, while still losing coverage compared to the unrestricted parser. EDM f-score is largely dominated by coverage, but not completely. For any configurations with similar coverage, those using TnT are always more accurate than C&C results. However, with any form of gap handling, TnT usually has lower coverage. This is interesting in view of the fact that, when no gap handling is used, TnT has higher coverage. A closer look at the differences shows that TnT makes more errors in closed class words. Since the unknown word handling mechanisms assume that closed class words will rarely be unknown, generic lexical entry coverage of these words is not as good.

In terms of parsing time, those results with the highest coverage show the approximate three-fold speed up predicted for **pos** tags in the earlier tables. For the $jhpstg_t$ data set, where the tagging is most accurate, the best f-score (from configuration TnT, assigning **pos**, with fallback to PTB POS tags) is only 0.056 below that when using no restriction. In certain circumstances, that might be an acceptable drop for the associated gain in speed. While parsing times do drop further for finer grained tags, since these decreases comes along with large drops in coverage, it is difficult to make any definitive claims about relative speed benefits.

Comparing the three gap handling strategies, we see that, in terms of coverage, PTB POS tags perform better than supertags, which are better than no gap handling. This is unsurprising and fits well with the results of Chapter 5. It is important to realise that in these experiments where every token is restricted, all lexical gaps are considered ‘caused by lexical restriction’, even when they would occur without restriction. Hence, with a gap handling strategy of *none*, parsing will fail if any words in the input are not in the lexicon. This is particularly clear on the *ws02* and *cb* data sets, which have a high percentage of unknown words. More appropriate baselines for these configurations would be those in Tables 3.3 and 3.5. Looking at time, we see that, as in Chapter 5, using supertags for gap handling is slower than using PTB POS tags, and this is particularly so for the fine-grained tags which have more associated generic lexical entries. Indeed, that problem is exacerbated here since fine-grained tags means both more lexical gaps and more generic entries for each gap. As mentioned above, this strategy could be fine-tuned to use less fine-grained tags for gap handling, but as long as the lexical restriction leads to a high percentage of gaps, any speed benefits of restriction may be lost in the slow-down caused by gap handling. Using single tagging, only **pos** and **pos+morph** tags can be predicted accurately enough to maintain a workable proportion of lexical gaps. The next section looks at one method to reduce the gaps caused by lexical restriction — allowing multiple supertags per token.

6.4.2 Multi-Tagging

In these experiments, a variable number of supertags is assigned to each token, depending on the tag probabilities and a β value that is varied between 0.0005 and 0.5. A large β value leads to fewer tokens having more than one tag. For instance, when β is 0.5, a token will only have multiple tags when the top tag has a probability below 67%, and not always then. For the TnT tagger, this leads to between 4% and 8% of tokens having multiple tags. As we saw in Chapter 4, the C&C tagger has a flatter probability distribution than TnT and so a β value of 0.5 can mean multiple tagging up to approximately 15% of tokens. Using multiple tags in this way is less restrictive than single tagging, and so leads to higher coverage, but without the same speed increase.

Again, many configurations were tested, with the same eight tag forms and two taggers. Since we saw in the previous section that using PTB POS tags for gap handling always gave higher coverage and speed than using supertags, without showing any decrease in accuracy, we use only the two gap handling strategies in this section — none and PTB. In the previous experiments, using no gap handling was always faster, but at the expense of a drop in coverage. In these experiments, by varying the β level, we are able to compare at equal coverage levels, and found that using PTB POS tags is always faster than no gap handling for the same coverage level. This results from the fact that, to get the same coverage level without gap handling, a much less restrictive β setting was required which assigned more supertags on average, hence allowing more lexical items into the chart.

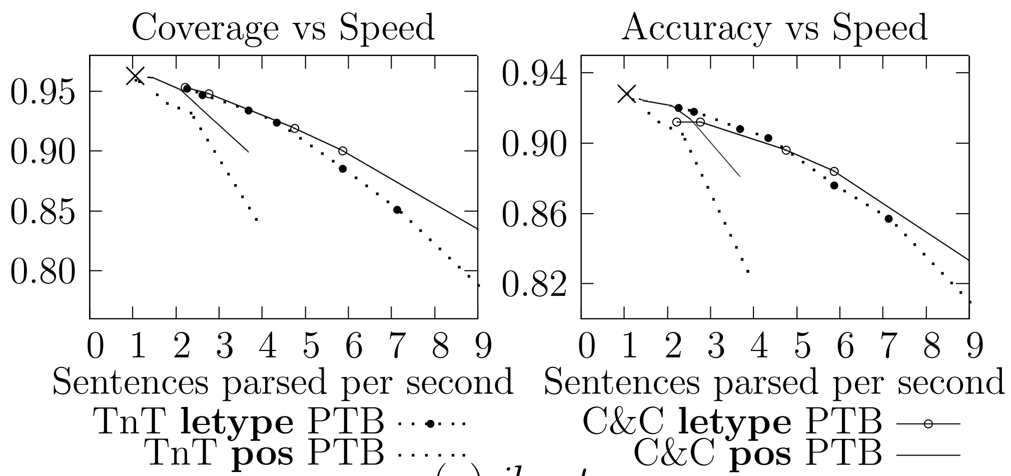
Comparing tag forms, there was very little difference, for either tagger, between **letype**, **letype+sel**, **letype+morph** and **letype+sel+morph**, with **letype** the best of the four. Comparing between **pos**, **subcat** and **letype**, we see that where the same level of coverage can be achieved, parsing speed increases as tags get more specific. However, when tagging with **letype**, we can not reach the same level of coverage as with **pos** tags, as sometimes the taggers are very certain of one tag, but wrong. Figure 6.2 shows how coverage and accuracy (measured as EDM f-score) vary according to sentence throughput, using the two best configurations for each tagger, where best is either fastest (and hence **letype**) or highest coverage (**pos**). All cases are using PTB POS for gap handling. We can see here that the C&C tagger generally gives the best results, although we see the same pattern as before in that where the coverage is similar between the taggers (**letype** over data sets $jhpstg_t$ and *cb*), TnT has higher accuracy. This points to another trend in tagger differences — the C&C tagger is more likely to assign a tag that allows a valid analysis, but is incorrect. While this is a difficult trend to quantify, it suggests that the C&C tagger optimises more for valid tag sequences, while TnT's decisions are more local.

Compared to parsing without restriction, we see that using **pos** tags we can achieve the equivalent coverage with a small or no decrease in accuracy, but only on the *ws02* data set do we see an increase in coverage due to less time outs. However at these coverage levels, the speed increases are minor. If accuracy decreases of between 0.007 and 0.018 are acceptable, however, speed can be doubled and a 5% or 6% loss in coverage can lead to parsing six times as many sentences per second.

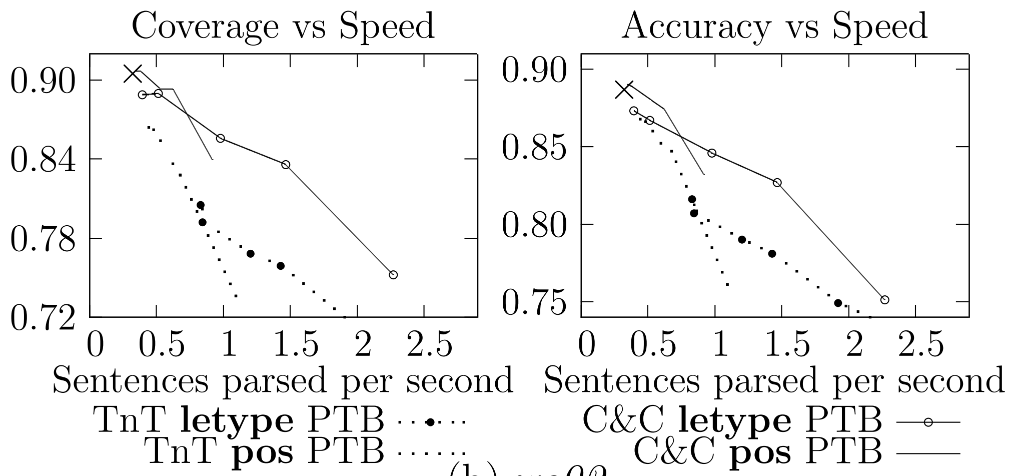
6.4.3 Selective Tagging

Multiple tagging increases coverage over single tagging, but at the expense of adding extra input tokens to the parsing chart. An alternative is to be more selective about when to potentially reduce coverage by lexical restriction. Here we assign the most likely supertag only if the tag probability is higher than a threshold value. Where the threshold value is low, the results of this method approach those from single tagging, but by increasing the threshold we allow a more nuanced restriction that can be adjusted depending on the requirements of coverage, accuracy and speed. We varied the threshold from 0.5 to 1.0, and again compared the results from the two taggers, eight tag forms and two gap handling strategies.

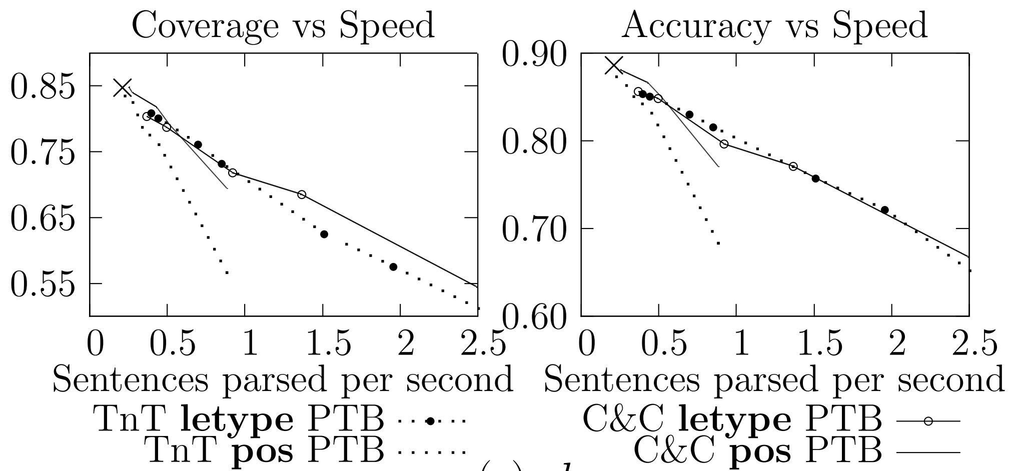
Using selective tagging in this threshold range, we found that C&C had clearly better performance. This is unsurprising, given



(a) *jhpstg*$_t$

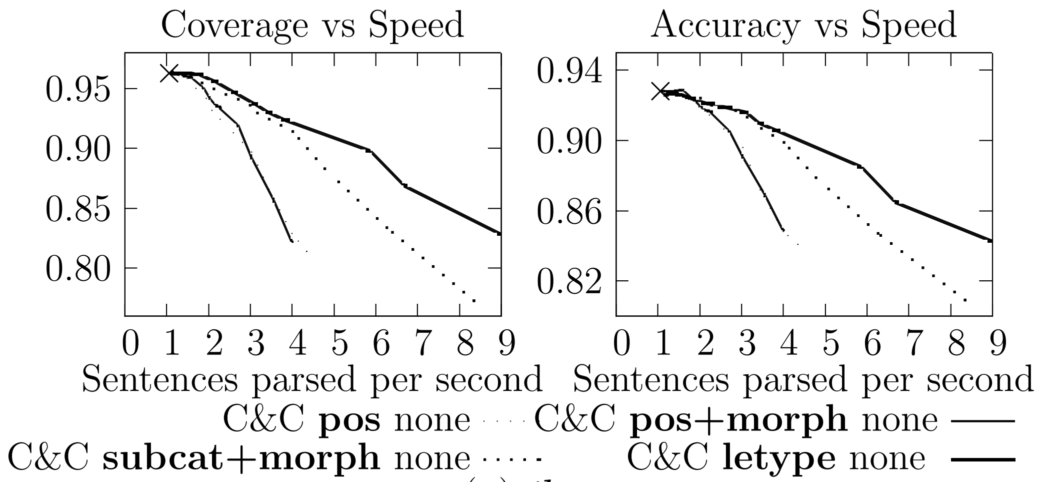


(b) *ws02*

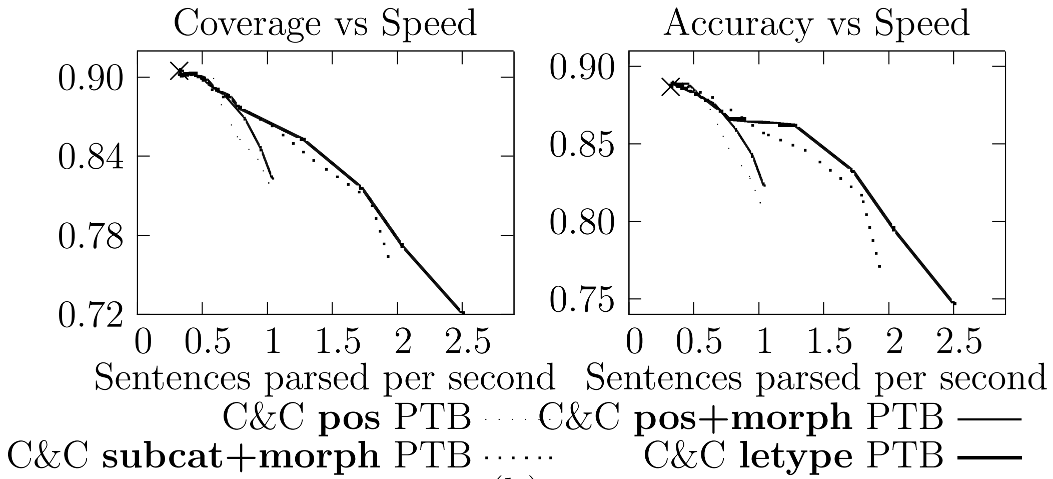


(c) *cb*

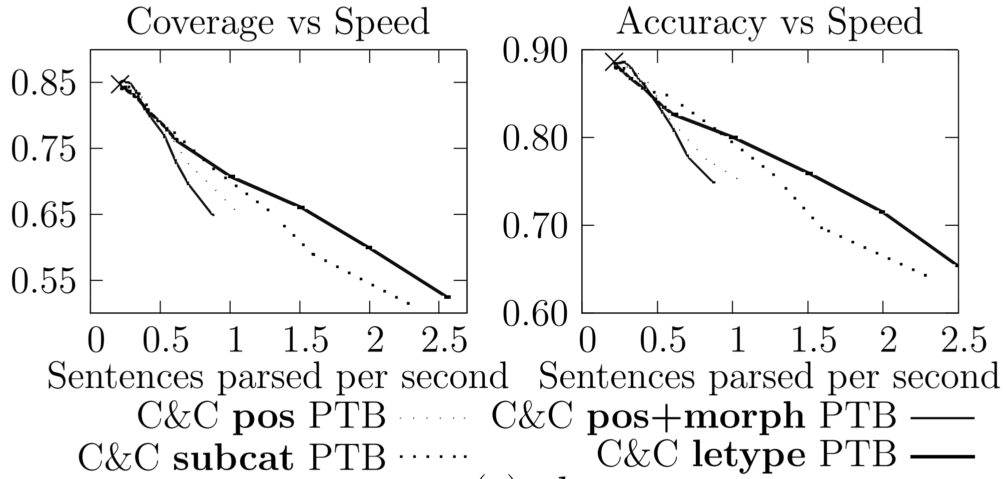
Figure 6.2: Coverage and accuracy (EDM f-score) versus sentence throughput using multiple tagging. The **letype** tag gives the best sentence throughput at almost every level of coverage, but the **pos** tags give the best absolute coverage. The relevant performance when no restriction is used is shown on each graph by $\times$.



(a) *jhpstg$_t$*

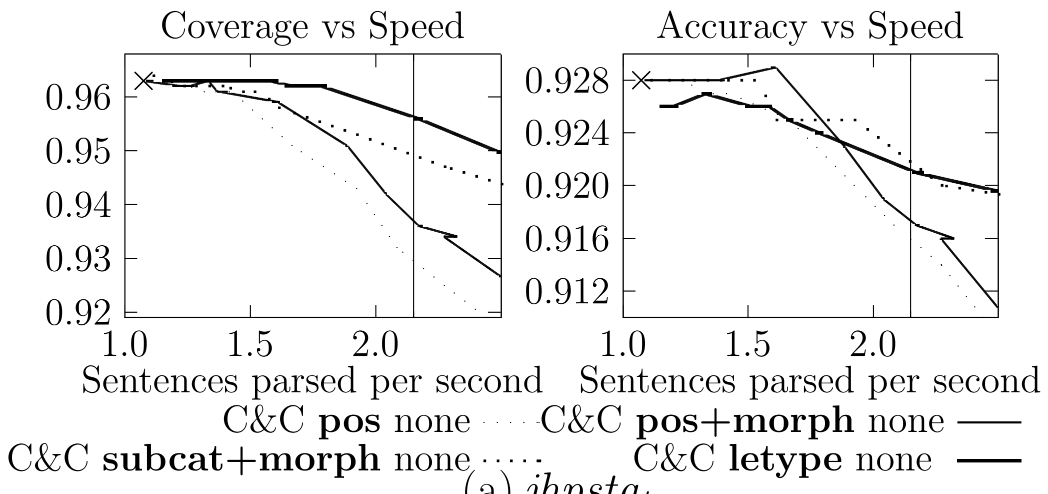


(b) *ws02*

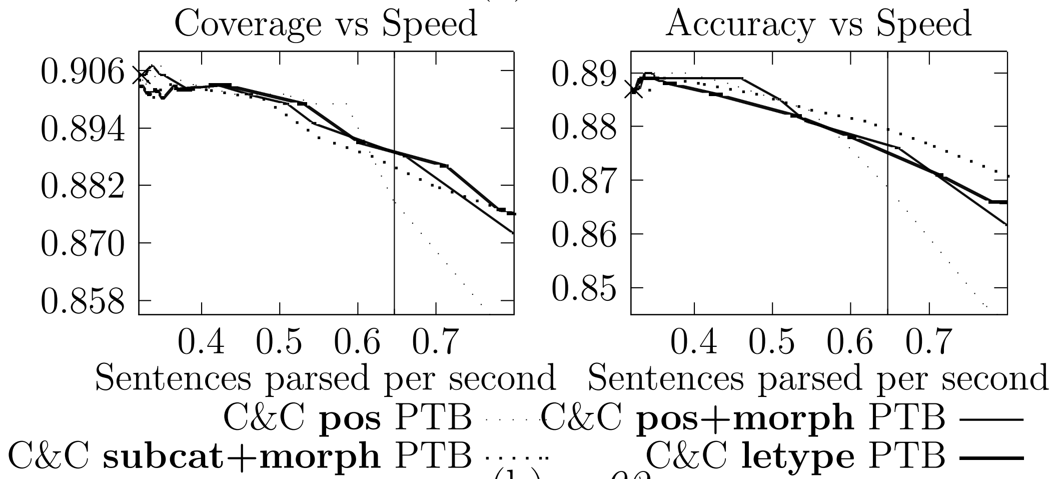


(c) *cb*

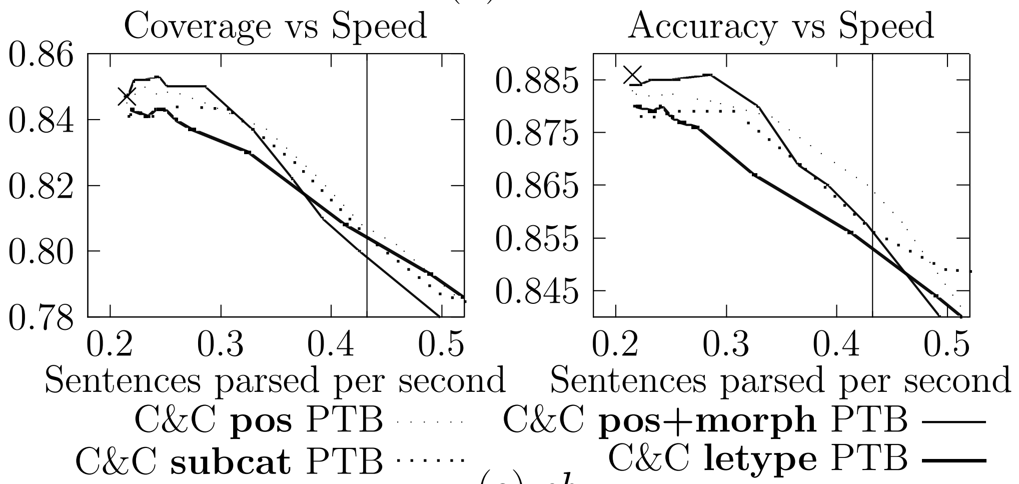
Figure 6.3: Coverage and accuracy (EDM f-score) versus sentence throughput using selective tagging. Curves are shown for the best performing configurations. The relevant performance when no restriction is used is shown on each graph by ×.



(a) *jhpstg*$_t$



(b) *ws02*



(c) *cb*

Figure 6.4: Close-up of coverage and accuracy (EDM f-score) versus sentence throughput using selective tagging. Curves are shown for the best performing configurations. The relevant performance when no restriction is used is shown on each graph by ×. The point where unrestricted speed is doubled is marked with a vertical line.

the results seen in Figures 4.11 and 4.12, where TnT performance was only superior when most of the tokens were being tagged. The better tag form depends on whether the emphasis is being placed on speed or coverage. Figure 6.3 shows, for each data set, configurations that, for some sentence throughput, had the highest coverage or accuracy. Each of these graphs suggests that **letype** assigned by C&C is the better configuration, at the maximum speed-up predicted using gold supertags. However, this speed up comes with a loss in coverage of up to 30%.

Compared to multiple tagging, selective tagging performs similarly when optimising for speed, however a closer look at the top of the graph shows where the benefits of selective tagging are. Figure 6.4 focusses on those threshold values that give up to twice the parsing speed, and in this range selective tagging shows better coverage and accuracy than multiple tagging. In terms of tag forms, at twice the parsing speed, **letype** still gives the best coverage, but the better accuracy comes when using **subcat** or **subcat+morph**, which appear to be a middle ground between speed and accuracy.

At this level, the better configurations over $jhpstg_t$ use no gap handling because the tag accuracy is high, and hence restriction causes very few gaps. For the other two data sets where tagging accuracy is not so high, and there are more unknown words, PTB POS gap handling is required to get good coverage.

Compared to no restriction, we can see that **pos+morph** tags can actually give an increase in coverage and accuracy, and a speed increase of 50% is possible for all data sets with no decrease in accuracy. At higher speeds, accuracy starts to decrease, and, as with multiple tagging, the highest speed possible depends on how large an accuracy decrease is acceptable.

6.5 Conclusion

Lexical restriction is a means of increasing parser efficiency by reducing the number of lexical entries in the parse chart. To see how effective the various supertag types from Chapter 4 could be at this task, we parsed using gold supertags. The results showed that, while all tag forms gave an increase in efficiency, the potential benefits were much less than those possible for CCG or LTAG parsing, owing to the fact that the lexicon already has a filtering effect on lexical entries.

Still, speed increases were possible, and so three different tagging strategies were tested, using supertags assigned by the TnT and C&C taggers. Single tagging, where each token is restricted to lexical entries compatible with the single most likely tag according to the tagger, yielded a three-fold speed up using **pos** tags, but at the cost of 8% in coverage on the data set where tagging is most accurate, and up to 20% for the *cb* data set, where tagger accuracy is lower.

When multiple tags could be assigned to each token (depending on relative tag probabilities), the same efficiency increase could be achieved with a smaller, though still noticeable drop in coverage. Here, we saw that where speed has a high priority, the **letype** tags yield a higher coverage and accuracy than other options at most levels of sentence throughput, but that **pos** still gives the best absolute coverage, at up to approximately twice the parsing speed. As discussed in earlier chapters, these coarse-grained tags would not, in general, be considered supertags since they don't explicitly contain dependency information. However, for this task, they provide the sort of information that is expected of a tagger and discard blatantly wrong analyses, while allowing the parser to make the fine distinctions. Furthermore, they are superior to Penn Treebank POS tags for this purpose, because they make exactly the distinctions that are made in the grammar, which the PTB POS tag set does not.

Using selective tagging, where a token is restricted only when the tag probability is over a certain threshold, we see that at a high level the results are very similar to multiple tagging. However, when focussing on the top part of the curve, the coverage decrease is more gradual. Speed increases of up to 50% were achieved with no decrease in coverage or accuracy, and in general selective tagging gave better results up to double the parsing speed.

Comparing between the TnT and C&C taggers, we found that despite TnT being the most accurate tagger when evaluating the top tag, there were three reasons that C&C was the better tagger to use in this task when coverage is important. Firstly, we saw in single tagging that TnT, while making fewer errors, made more errors in closed class tags where gap handling is not as effective. Secondly, as we saw in Chapter 4, characteristics of C&C's probability model mean that it is more accurate when multiple tags are assigned according to the probabilities of this model. Finally, although it is difficult to quantify exactly, results suggest that C&C optimises more for a valid sequence that will lead to a parse, compared to the more local optimisations of TnT, which lead to more accurate tagging, but fewer parseable tag sequences.

The final results show that modest efficiency increases are possible using lexical restriction. When speed has high priority, the **letype** tag form gives the best coverage and accuracy at higher speeds, but when maintaining high coverage is important the **pos** and **posmorph** tags, using selective tagging can give an increase in speed of between 50% and 100%.

7 Parse Ranking

Parse accuracy, when evaluating the top ranked parse, is affected both by the grammar, and by the statistical model that ranks the possible analyses that the grammar can produce. In previous chapters, we have ignored the effect of the statistical ranking model and measured relative accuracy differences by modifying the grammar and holding the ranking model fixed. We made the simplifying assumption that adding (Chapter 5) and removing (Chapter 6) possible analyses did not change the ranking of unaffected analyses, in order to focus on the effects of various grammar modifications associated with using supertag information. In this chapter we look at using the supertag information in a different way — to inform the parse ranking stage.

While we saw that lexical restriction can have an effect on parse accuracy, there is a fundamental difference in how supertags were used in those experiments, and how they will be included in parse ranking. In lexical restriction, supertags form hard constraints on the solution of the parsing problem. Even with multiple tagging or selective tagging, the constraints, while more relaxed, are fixed and no analysis that does not satisfy them will be produced. In contrast, for parse ranking, supertags will be used as *soft constraints*. In constraint programming, soft constraints define preferences regarding the optimal solution and are usually encoded in the form of a cost function to be maximised or minimised.

There are many ways that lexical statistics could be added to the statistical parse ranking model, and exploring the full spectrum of possibilities is out of scope for this thesis. The aim of this chapter is to explore the space of possibilities and describe some preliminary experiments designed to investigate how much information is available in the tagger models we have trained.

7.1 Soft Constraints

Soft constraints were originally a solution to over-constrained problems in the field of constraint programming. Later, it became obvious that framing a problem as one of optimising soft constraints, rather than satisfying hard constraints was also suitable for modelling systems that included some concept of fuzziness, preference or probability. Parsing natural language fits both scenarios. When we use lexical information to form hard constraints, such as enforcing that a token must have a type compatible with a, we run into robustness issues if no solution can be found to satisfy that constraint, an instance of over-constraint. As discussed in Chapter 3, this can occur either because of something missing in the grammar, or because of ungrammatical input, but in either case leads to parse failure. In previous chapters we looked at solutions to this problem. The ideal solution is to predict the correct tag in the first place, using a more accurate tagger trained on additional training data. In a real, non-ideal situation different strategies for assigning multiple tags can be used, and so we can allow types compatible with a, b or c, in constraint terms, enlarging the domain of a variable. But what about d? The more options we give, the greater the ambiguity in parsing.

There are two issues arising from greater parsing ambiguity. The first is the higher number of possible analyses from which to select the "correct" analysis. This is where the idea of using soft constraints for modelling systems with preferences or probabilities comes in to play. Semiring-based constraint satisfaction (Bistarelli *et al.* 1997) is a framework for assigning preference weights to different values a variable might take, and defining the means by which these preferences weights are combined to determine the 'best' solution. The preference weights are in the range $[0, 1]$, where a weight of zero or one is a hard constraint, representing total incompatibility or absolute necessity of a value respectively. In the case of a PCFG parser, at each decision point (node), the set of values is the set of expansions that are possible for that node, and the preference weights are the probabilities associated with each expansion rule. In general, in a PCFG, all constraints are soft constraints, hence there are no robustness issues caused by over-constraint and the most likely parse is the one where the combination of probabilities was highest.

Given that robustness is a continuing problem in HPSG parsing, why do we not use soft constraints throughout the parsing process, as in PCFG parsing? This goes back to the second issue arising from greater parsing ambiguity: loss of efficiency. By adding more and more possible values at each decision point, the parser has to make more choices, leading to slower parsing times and greater memory use. In a PCFG parser, this problem is mitigated by two factors. Firstly, the analysis is not as detailed as in HPSG, and so there are much fewer possible values to consider. Secondly, because each subtree is considered independent, the parsing chart can be pruned at each node, keeping only those sub-analyses that yield the highest probability while still being able to find the best full analysis, according to the PCFG model. Even given these factors, reported PCFG parsing times are not significantly better than those we reported in Chapter 6. It might still be informative to create a PCFG by multiplying out all observed feature value combinations from a treebank created with a broad-coverage HPSG-based grammar such as the ERG. However, while this could provide interesting observations about the amount of information in such a treebank, it is doubtful that there is sufficient training data available to provide accurate probabilities for such a grammar.

Returning to our recurring theme of trade-offs, the balance between hard and soft constraints reflects the trade-off between robustness and efficiency. There are many ways that the different types of constraints could be mixed during parsing to affect this trade-off. At the present point in time, the PET parser combines the two in a fairly rigid manner, using hard constraints while filling the parsing chart, but employing soft constraints in the later ranking stage, the stage where the probabilities and preferences of natural language are best balanced. There is certainly scope for introducing soft constraints in the first stage of parsing, without going all the way towards a PCFG. However, in this work we will focus on possible improvements to the current utilisation of soft constraints, using the lexical statistics that have been learnt by the supertaggers.

7.2 The Lexical Statistics

There are various ways of using the lexical information that the taggers can give us. In this preliminary investigation, we will focus on two statistics that represent different types of information. Both are global statistics, based on the full tag sequence extracted from each potential parse. These tag sequences are extracted using exactly the same method as was used to extract the training data for the supertaggers, which is described in Chapter 4.

The first lexical statistics feature uses the probabilistic models created for the supertaggers to calculate the tag sequence probability. In this scenario, we want the sequence probability not for the most likely sequence, but for an arbitrary tag sequence as extracted from an analysis. The supertaggers do not, by default, output the sequence probability, even of the best sequence they produce. They do, however, output the marginal probabilities of the tags they predict and we use these to calculate an approximation of the tag sequence probability. In order to calculate this probability according to the tagger's model, we ran each tagger on each test set with a β value of 0.000000001 and recorded the tags and their associated probabilities according to each model. Then, for each sentence in the test data, we extracted the tag sequences from each of the analyses returned by the parser, up to 500 analyses, and looked up the probabilities given by the model for each tag. The tag sequence probability then was the product of the individual tag probabilities. No additional smoothing was used, as we were interested in the probabilities as given by the models. Since both C&C and TnT restrict the tags that can be seen with particular word forms, this can lead to a tag probability, and hence a sequence probability, of zero. This is something that could be fine-tuned in future parse ranking experiments.

Since we have seen, with the TnT tagger, that even when the probabilistic model isn't helpful for our task, single tag accuracy can be good, we also look at a second statistic more aligned with single tag accuracy. This second lexical statistics feature ignores the tag probabilities and measures the compatibility between the tag in the parse and that predicted by the supertaggers. The value used here is the percentage of tokens in each parse that were compatible with the predicted tags. This type of feature could be fine-tuned to ignore certain word or tag types (such as closed class tags or named entities), or conversely to put greater weight on getting particular token classes correct. For the purpose of this investigation, we simply align the tokens and tags as the parser would in lexical restriction, but more intelligent handling of multiword expressions would also be an area for future improvements.

There are other possibilities for lexical statistics features, including merging different tag granularities, instead of trying to determine the most appropriate, or perhaps trying to incorporate some measure of tagger confidence learnt in training, at a global level or for particular tags. In the following section however, we will look at how the basic statistics described above could be used to improve parse ranking.

7.3 Incorporating Lexical Statistics in Parse Ranking

Lexical statistics can be added to the parse ranking stage of parsing in a variety of ways that vary in their complexity and speed of implementation. The simplest methods involve reranking the output of the current statistical model. Both Collins (2000) and Charniak and Johnson (2005) have shown that reranking the output of a statistical parser can bring significant improvements in accuracy, but the common reason given for reranking in this scenario is to add features that are difficult to add to a generative statistical model. We already use a discriminative model that can incorporate arbitrary global features, although currently no lexical features are used. However, reranking could also been seen as a simple way of adding additional information without needing to train a full model.

7.3.1 Reranking

Taking the top 500 parses as returned when using the ERG's default Maximum Entropy statistical model, we looked at what information the tagger models could contribute towards selecting the best parse. The first step was to look at the discriminative power of the two lexical statistics, as provided by the different

	Original Model			Reduced Set		
Data Set	Average	Median	Precision	Average	Median	Precision
jhpstg$_t$	191.34	56	0.135	26.50	6	0.265
ws02	240.89	119	0.142	36.97	7	0.224
cb	283.65	424	0.091	27.26	8	0.165

Table 7.1: Average and median number of parses as produced by the original model (limited to top 500) and in the reduced set obtained by looking at only the most likely tag sequences. Precision is calculated by randomly selecting an analysis within each set, and evaluating in terms of exact match against the gold standard.

tagging models. Dalrymple (2006) reports a similar investigation using gold standard LFG tags from the PARGRAM grammar, however she only has a small data set which has gold standard annotation. Toutanova *et al.* (2002) discuss using tagger output to discriminate between parses from an earlier version of the ERG, using very fine-grained tags. Despite the differences in tag granularity and parser formalisms, both find that determining the gold standard sequence of tags leads to a 50% reduction in ambiguity. In this experiment we don't use gold standard tags, but only look at the parses that were most likely, according to the lexical statistics provided by the taggers. Table 7.1 shows the best result obtainable by using the available lexical statistics. Here we see it is possible to reduce the set of parses down to 10%-15% of the original set. Of course, not all of these sets will now contain the correct parse. To get some idea of how this reduction could affect the accuracy, for each item we randomly selected a parse from the set and evaluated using exact match. Repeating this ten times gave us an average precision. In all cases, this is substantially better than randomly selecting from the entire set, despite the fact that for many items there was no chance of selecting the right parse.

The effect of random selection to a large extent drowns out the

Data Set	MaxEnt Model Rank	Sequence Probability	Tag Compatibility
jhpstg$_t$	0.459	0.470	0.484
ws02	0.375	0.381	0.385
cb	0.265	0.228	0.228

Table 7.2: Best exact match precision when using MaxEnt model rank, tag sequence probability or tag compatibility scores as the first ranking criteria, backing off to the other criteria as needed to narrow the set of parses down to one.

differences between taggers, tag forms and lexical statistics, with a preference to fine-grained tags the only trend strong enough to show up. Looking at the reduced sets as a whole, we can see that tag compatibility in most cases leads to smaller sets. For the *ws02* and *cb* data sets, this goes hand in hand with a lower percentage of items which have the correct parse in the reduced set, but in *jhpstg*$_t$, where the tagger models are more accurate, we see smaller sets that are *more* likely to contain the correct parse.

As an alternative to randomly selecting a parse from within a reduced set, we looked at a number of ways of combining the information that we have to select the one best parse. The first method naively combines the sequence probability, tag compatibility and original parse rank from the statistical model by using them as distinct ranking criteria, ranking first by one and then backing off to the others if there is still more than one parse with the same ranking. Since the original rank uniquely identifies a parse, it is always the last criteria and we varied which of sequence probability and tag compatibility to use first. Table 7.2 shows the best results when varying the first ranking criteria.

For the *jhpstg*$_t$ data set, where there has been sufficient in-domain training data for the taggers to achieve reasonable accuracy, the best results came when using the most fine-grained tags (**letype+sel+morph**) as predicted by the C&C tagger. Over the *ws02* data set, where the C&C tagger never achieved the same level of accuracy as TnT, the better results all come from TnT, although the differences between the tag types, and indeed between the three ranking criteria are small, with tag compatibility against the **pos+morph** tags giving slightly better results than any other configuration. The lower tagger accuracy for the *cb* data set means that there was no apparent benefit to using the lexical information in this sort of model, and any differences between taggers and tag forms appeared random.

The most interesting observation from these results is that, when looking at the $jhpstg_t$ data set which is most representative of the benefits accurate tagging could bring, the **+morph** tag variants are the ones that improve over the original ranking. This differs from the earlier tasks, where morphology information was found to add very little, except to the coarse-grained tags. Here we see over 3% improvement in accuracy between using **letype+sel+morph** and **letype+sel** tags. This suggests that when we look at less naive methods of adding lexical information, we should perhaps focus on predicting morphological information.

A slightly less naive method of using the information is to calculate a score from a linear combination of sequence probability, tag compatibility and MaxEnt score (rather than rank). Given no extra information, the easiest combination is to give all three criteria equal weighting. However, there is some training data available that allows us to learn weights for each criteria. This is the same tourism data that was used to train the original MaxEnt model, and so may overfit to that domain of data. Table 7.3 shows the best results using an even weighting of criteria and also the weighting learnt for each tagger and tag form combina-

Data Set	Equal Weighting	Learnt Weighting
jhpstg$_t$	0.493	0.494
ws02	0.392	0.394
cb	0.230	0.234

Table 7.3: Exact match precision when using a linear combination of MaxEnt model score, tag sequence probability and tag compatibility scores to rank parses, first with each criteria given equal weighting, and then with weightings learnt from tourism training data.

tion using the extra tourism data.

We see here that learning the weights gives only a very slight, statistically insignificant improvement over just giving each criteria equal weight, despite the fact that the learnt weights are not similar. For all tagger and tag form combinations, sequence probability is given a much lower weight than either of the other criteria, with tag compatibility in most, but not all, cases given the heaviest weight of the three. For the *jhpstg*$_t$ and *ws02*, both combinations give a significant improvement over all the ranking criteria in Table 7.2, but again we see that the lexical information we have brings no improvement over the original ranking for the *cb* data set, and the differences between the different methods for using this information are insignificant for this data set. Echoing the results from Table 7.2, the best tag forms for the *jhpstg*$_t$ data set are the fine-grained tags with morphological information from the C&C tagger, while tags from TnT give the best results over *ws02*, with most tag forms (except **subcat** and **pos**) producing very similar results.

7.3.2 Expanding the Maximum Entropy Model

Reranking is a simple way of giving us some indication of how the lexical statistics may contribute to parse ranking. In this section we outline possible future experiments that incorporate these statistics into the Maximum Entropy (MaxEnt) model.

The simplest method would be to treat the two statistics used above as *auxiliary distributions* (Johnson and Riezler 2000). Generally, an auxiliary distribution is a probability distribution estimated from a larger, simpler corpus than that used to train the main model. The log probability according to the auxiliary distribution is added as a feature to the full model. Auxiliary distributions were proposed as a method to include lexical selectional preferences in parse disambiguation, and have also been tried as a means of domain adaptation (Plank and van Noord 2008). So far, auxiliary distributions have not been proven to bring much, if any improvement, over a base model in parse disambiguation. It is not clear why adding extra information degrades performance. One possibility is that the simpler structures learnt in the auxiliary distribution are not a good match for the structures in the grammars. In our situation, there is no mis-match between structures, and since we have already seen that the lexical statistics above brought some improvement in a combination model, it would be worthwhile to see if inclusion in the full model could improve over that.

The general theory suggests that auxiliary distributions should be based on annotated corpora that are larger and simpler than the main training data since they bring more data for less cost, but there is minimal difference between the effort involved in producing training information for the Maximum Entropy model, and that for the supertagger models. It is true that the supertagger only uses the tag sequence without the full parse structure, but assigning the tag sequences, particularly with the fine-grained tags, is very little easier than assigning the full analysis. So what is the point of learning them separately? One possible answer to that considers the *quality* of the training data. We saw in Chapter 4 that the tag accuracy of the unannotated parser output was quite good, even on the Wall Street Journal data, where parse accuracy is much lower. Self-training (i.e. training a parse model on unannotated parser output) has been shown to be effective for improving accuracy in a PCFG parser (McClosky *et al.* 2006a), but hasn't been attempted for the PET parser. An interesting experiment would be to compare between ranking models that have fully trained with unannotated parser output and those that have been enhanced with lexical statistics from a supertagger trained on parser output (as in Chapter 4). The hypothesis being that the constrained nature of the grammar is enough to produce good quality, internally-consistent tag sequences within the size of the context window used by the taggers, whereas training the parsing model on parser output will reinforce bad ranking decisions.

Another possible direction to consider for introducing lexical statistics to the Maximum Entropy model was touched on in Section 7.2. The statistics we have used so far have been simple global summaries of the tag sequence, but more fine-tuned features could be included. Rather than have tag compatibility, for instance, be a percentage across the whole sentence, individual features could count aspects like *the percentage of tokens classed as noun types where the tagger predicted a verb*, or, given the positive effect that morphological predictions seem to have, we could count the number of times that a token type was compatible with the predicted **pos** but not the predicted **pos+morph** tag (or the equivalent at different levels of granularity). The possible feature space in this regard is huge, but at least the preliminary experiments we have conducted here give some indication of which information is useful and predictable. A more comprehensive error analysis of tagging accuracy could also direct the feature engineering.

7.4 Conclusion

Modelling a system in terms of soft constraints is useful when hard constraints would regularly lead to no solutions, or when the system being modelled includes ideas of fuzziness, preference or probability. Natural language parsing falls into both categories. This chapter focusses on using lexical statistics as soft constraints for the second purpose, introducing lexical preferences into the ranking stage of parsing.

We experimented with two different types of lexical statistics based on the tag sequences extracted from possible analyses: the first, sequence probability, makes use of the probabilistic models learnt by the taggers, while the second, tag compatibility, looks only at the tag instances that are predicted as most likely. Looking at the discriminative power of these statistics, we found that taking only those parses that were most likely according to each statistic can reduce the average number of analyses down to 10%–15% of what the parser returns (when limited to 500 parses). Selecting randomly from this reduced set gave better exact match precision results than selecting from the full set, despite the fact that the reduced set was not guaranteed to contain the correct parse. Looking at the reduced sets themselves, without introducing random selection, we found that tag compatibility produced smaller sets of possible analyses and, on the in-domain test set where the tagging was most accurate, these smaller sets were more likely to contain the correct result than those distinguished by sequence probability.

Moving on to non-random selection methods, we experimented with different methods of combining sequence probability, tag compatibility and the original Maximum Entropy score. The first was as a very simple set of ranking criteria, backing off from one criteria to the next as necessary to narrow the possible parses down to one. Next we tried calculating a ranking score using a linear combination of the three criteria, first with each having equal weighting, and then by learning the optimal weights using additional training data in the tourism domain. We found that learning the weights gave only a slight, statistically insignificant improvement over equal weighting, despite the learnt optimal weights having a repeatable, non-equal pattern across tag forms. All methods of combining lexical statistics with the original Maximum Entropy ranking improved over the original ranking on the $jhpstg_t$ and *ws02* data sets, but the lower tag accuracy achievable on the *cb* data set meant that the lexical statistics there added only random noise. Where the lexical statistics were beneficial, we saw that morphological information, unlike in previous chapters, gave a significant boost to results. For the in-domain $jhpstg_t$ data set, where there is enough relevant training data to start forming an accurate model, the fine-grained tags as predicted by the C&C tagger yielded the greatest improvements, but we also saw that tags from TnT can give some advantage in situations where less training data is available.

The above observations are meant to provide a starting point for selecting features to add to the Maximum Entropy model. Section 7.3.2 gave some suggestions for future work in this direction.

8 Conclusion

Deep hand-crafted grammars can produce very detailed syntactic and semantic analyses of natural language that can be useful for applications such as information retrieval and machine translation. The aim of this thesis was to discover how lexical statistics could be used to improve the performance of parser that uses such a grammar. The result has been a story of trade-offs. We evaluated the PET parser, together with the broad-coverage HPSG-based English Resource Grammar, in terms of robustness, efficiency and accuracy. Each set of experiments that we carried out painted a complex picture showing how to improve one factor at the cost of reduction in another, but also described how best to configure the system to balance the three factors, or to give priority to one or two of the three.

Chapter 2 outlined the respective advantages and disadvantages of deep hand-crafted grammars, as opposed to other systems that are derived from treebanks or that provide a less detailed analysis. We found that the many of the sources of disadvantage in the deep hand-crafted system were also the source of the system's advantages. Rather than eliminating the advantages in efforts to ameliorate disadvantages, we look to combine advantages from different systems. Hybrid processing is one means by which to accomplish that and we focus on *supertagging* as a particular hybrid processing technique that has been shown to work well for a lexicalised formalism such as HPSG.

In Chapter 3 we discussed efficiency, robustness and accuracy, the three often competing factors used to evaluate parser performance. In particular, we looked at the problem of evaluating and comparing accuracy between different parsers, and for different purposes, finding that much of the confusion surrounding parser accuracy metrics stems from an ill-defined purpose both of parsing and of evaluation. Learning from this, we define our parsing goal as producing semantic information (in terms of *who*, *what*, *when*, *where*, *why* and *how*) for an application, and further specify that our evaluation is aimed at finding the optimal configuration of the PET parser together with the ERG for that task. The major contribution of this chapter is a new granular evaluation metric, Elementary Dependency Match (EDM), that is suitable for evaluating the detailed semantic information that is produced by the PET parser.

The lexical statistics that we experimented with were those learnt in the process of training tagger models for a variety of different tag forms. The tag forms varied in granularity and type of information, from the **pos** tags, which are make only very broad distinctions, up to the **letype+sel+morph**, which include information about morphology, subcategorisation and selectional preference of prepositions. There were two questions we were interested in with regards to these tags:

(a) Which tag forms are predictable from the available training data?

(b) Which tag forms are useful for different aspects of the parsing process?

In order to answer the first question we ran a set of experiments comparing the performance of TnT, a Hidden Markov Model-based POS tagger, and the C&C supertagger, a Maximum Entropy Markov Model-based tagger. The experiments varied the amount and source of the training data, and also the method by which tags were assigned. Results from these experiments showed that the simpler TnT tagger achieves better results on less training data, when evaluating only the single best tag predicted by the tagger. However, if sufficient in-domain training data is available, or when the method of assigning tags makes use of the underlying probability distribution model learnt by the tagger, the C&C supertagger has superior performance.

To explore the usefulness of the different tag forms, we looked at three different aspects of the parsing process where lexical information has been demonstrated to help. One of the main sources of parser failure is unknown words in the input. Hence, the first set of experiments used lexical statistics to predict information about an unknown word, in order to boost parser robustness. Here we used four different levels of granularity related to the word class and its subcategorisation, but no morphology information. In addition to the tag forms based on the types of the grammar, we also used Penn Treebank-style POS tags predicted by a tagger trained on one million words. These Penn Treebank POS tags have been the parser's default method of injecting external information where the grammar needs supplementation, and part of this investigation was to quantify the effect the tags have. It had been suggested that using more detailed tags, or tags more aligned with the grammar would lead to better quality analyses. The results showed that it was possible to get more precise analyses with the **letype** tags, but less of them. That is, in the three-way evaluation trade-off, more detailed tags brought greater precision, but lower robustness than the Penn Treebank tags. In comparison to using no external information, the **letype** tags provide a good option when precision is paramount, but when balance between efficiency, robustness and accuracy is required, the default tags are a good compromise.

The second aspect of parsing that we considered was the impact of lexical ambiguity on parser efficiency. This has been the focus of previous work involving supertags, and following that work, we use the predicted tags of different forms to restrict the lexical items that are considered in parsing. The first stage was an oracle experiment with each tag type, in order to determine the upper bound on performance possible by using this technique. We found that the massive potential for improvement previously reported for CCG and LTAG parsing was not possible in our situation. The potential space for improvement is narrowed from both the top and the bottom. To start with, the baseline is not as low as the other frameworks. The PET parser, without any tuning, can already parse sentences at an acceptable speed, though not the blinding speeds of the C&C parser. Secondly, the level of detail produced is greater which *may* mean that the top speed is limited without making approximations. Despite these caveats, we found that there was space for improvement and so a set of lexical restriction experiments were conducted.

Three different methods of lexical restriction were tested: first, allowing only lexical items compatible with the single top tag predicted for each token; then allowing multiple tags, depending on the probabilities assigned by the taggers to the tags; finally selectively restricting particular tokens of the input, depending on the probability assigned by the tagger to the top tag. Again, we were mapping the trade-off space. The clearest result was that a speed increase of 1.5–2 times the base speed was possible without losing accuracy or robustness. This result was achieved using the basic **pos** tags, a tag form that would not fit any standard definition of supertag, since it represents *less* information than most POS tags. Not only does it not encode subcategorisation information, but even the morphological distinctions that are made in Penn Treebank POS tags are absent. Looking at the more complex picture, we found that where speed has priority, using the **letype** tag to restrict lexical entries leads to slower degradation in coverage and accuracy as speed increases.

Finally, lexical statistics have also been shown to be useful for increasing the accuracy of parse ranking. We looked at the potential benefits of using the lexical information encoded in our tagging models as soft rather than hard constraints, by using them in the parse ranking stage. Some basic preliminary reranking experiments showed that, when tagging was accurate, the fine-grained tags as predicted by the C&C tagger brought some improvement over the default statistical parsing model that is released with the ERG. In contrast to the earlier experiments, in parse ranking it appears that information about predicted morphology is brings a significant advantage. Full inclusion of the lexical statistics into the Maximum Entropy model was beyond the scope of the current work, but we outlined a number of steps that could be taken in this direction in future work.

8.1 Extensions and Future Research

Each set of experiments we carried out left open questions and room for further experimentation. Here we list some of the potentially fruitful areas for future research.

Evaluation The new EDM metric satisfied the criteria required for the current work, but there are many remaining questions regarding its general applicability, customisation and comparison to other evaluation metrics. Since one motivation for a granular evaluation based on 'semantic' units is that is better reflects human preference between better and worse analyses. This hypothesis could be tested in a human evaluation experiment by collecting pair-wise preferences from a human annotator and comparing the relative rankings with EDM scores. This could also give some indication of the relative importance of the different triple types.

There are also interesting possibilities in using the EDM metric for slightly different purposes. Since the Elementary Dependencies are closely aligned to the underlying grammar, there are potential benefits to the grammar writer in using EDM for error detection and classification. EDM could also be used where sentence similarity was required.

Learning Supertags The best results in learning supertags came from two sources: domain adaptation and training on parser output. The obvious next step in this direction is to combine these two. Small additional amounts of gold standard Wikipedia data yielded significant accuracy increases on Wikipedia test data. Since there is a large amount of Wikipedia data available, this would be the perfect source for uncorrected parsed data.

Unknown Word Handling Since we found the generic entries based on lexical types could give more precise analyses than Penn Treebank POS tags, at only a small cost in coverage, it seems worthwhile to fine-tune these generic entries in an attempt to bring the coverage to an equivalent level. To make this a viable unknown word handling strategy, some effort needs to be made to improve the efficiency of the generic entry instantiation in the PET parser. This would enable more flexibility in the lexical filtering rules, without the loss of efficiency that gap prevention currently brings.

Lexical Restriction A more sophisticated approach to lexical restriction would require substantial changes to the parser implementation, tying the supertagger and parser closer together as in the C&C parser. This reworking could allow the parser to be more reactive, making adjustments to entries in the chart in cases of parse failure. Another area for more sophisticated processing is in gap handling. In our experiments, using normal unknown word handling turned out the best option of those we tried, but there are other possibilities, including using the supertag to filter incompatible POS tags, or using underspecified versions of the supertags when a lexical gap is created.

Bibliography

ABNEY, STEVEN. 1997. Stochastic attribute-value grammars. *Computational Linguistics* 23(4):597–618.

ADOLPHS, PETER, STEPHAN OEPEN, ULRICH CALLMEIER, BERTHOLD CRYSMANN, DAN FLICKINGER, and BERND KIEFER. 2008. Some fine points of hybrid natural language parsing. In *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC 2008)*, Marrakech, Morocco.

BALDWIN, TIMOTHY. 2005. Bootstrapping deep lexical resources: Resources for courses. In *Proceedings of the ACL-SIGLEX 2005 Workshop on Deep Lexical Acquisition*, pages 67–76, Ann Arbor, USA.

BALDWIN, TIMOTHY, JOHN BEAVERS, EMILY M. BENDER, DAN FLICKINGER, ARA KIM, and STEPHAN OEPEN. 2005. Beauty and the beast: What running a broad-coverage precision grammar over the BNC taught us about the grammar — and the corpus. In *Linguistic Evidence: Empirical, Theoretical, and Computational Perspectives*, ed. by Stephan Kepser and Marga Reis. Berlin: Mouton de Gruyter.

BANGALORE, SRINIVAS, and ARAVIND K. JOSHI. 1994. Disambiguation of super parts of speech (or supertags): Almost parsing. In *Proceedings of the 15th International Conference on Computational Linguistics (COLING 1994)*, pages 154–160, Kyoto, Japan.

BANGALORE, SRINIVAS, and ARAVIND K. JOSHI. 1999. Supertagging: an approach to almost parsing. *Computational Linguistics* 25(2):237–265.

BENDER, EMILY M. 2008. Evaluating a crosslinguistic grammar resource: A case study of Wambaya. In *Proceedings of the 46th Annual Meeting of the ACL*, pages 977–985, Columbus, USA.

BENDER, EMILY M., DAN FLICKINGER, and STEPHAN OEPEN. 2002. 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.

BENDER, EMILY M., DAN FLICKINGER, STEPHAN OEPEN, ANNEMARIE WALSH, and TIM BALDWIN. 2004. Arboretum: Using a precision grammar for grammar checking in CALL. In *Proceedings of the InSTIL/ICALL Symposium: NLP and Speech Technologies in Advanced Language Learning Systems*, pages 83–86, Venice, Italy.

BISTARELLI, STEFANO, UGO MONTANARI, and FRANCESCA ROSSI. 1997. Semiring-based constraint satisfaction and optimization. *Journal of the Association for Computing Machinery* 44(2):236–265.

BLACK, EZRA, STEVE ABNEY, DAN FLICKINGER, CLAUDIA GDANIEC, RALPH GRISHMAN, PHIL HARRISON, DON HINDLE, ROBERT INGRIA, FRED JELINEK, J. KLAVANS, MARK LIBERMAN, MITCH MARCUS, S. ROUKOS, B. SANTORINI, and TOMEK STRZALKOWSKI. 1991. A procedure for quantitatively comparing the syntactic coverage of English grammars. In *Proceedings of the Workshop on Speech and Natural Language*, pages 306–311, Pacific Grove, USA.

Blum, Avrim, and Tom Mitchell. 1998. Combining labelled and unlabelled data with co-training. In *Proceedings of the 11th Annual Conference on Computational Learning Theory*, pages 92–100, Madison, USA.

Blunsom, Philip, 2007. *Structured Classification for Multilingual Natural Language Processing*. Department of Computer Science and Software Engineering, the University of Melbourne dissertation.

Brants, Thorsten. 1997. Internal and external tagsets in part-of-speech tagging. In *Proceedings of Eurospeech 97*, pages 2787–2790, Rhodes, Greece.

Brants, Thorsten, 2000a. *TnT — A Statistical Part-of-Speech Tagger*. Saarland University, version 2.2 edition.

Brants, Thorsten. 2000b. TnT — a statistical part-of-speech tagger. In *Proceedings of the Sixth Conference on Applied Natural Language Processing ANLP-2000*, pages 224–231, Seattle, USA.

Briscoe, Ted, and John Carroll. 1993. Generalised probabilistic LR parsing for unification-based grammars. *Computational Linguistics* 19(1):26–60.

Briscoe, Ted, and John Carroll. 2006. Evaluating the accuracy of an unlexicalised statistical parser on the PARC DepBank. In *Proceedings of the 44th Annual Meeting of the ACL and the 21st International Conference on Computational Linguistics*, pages 41–48, Sydney, Australia.

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

BURKE, MICHAEL, AOIFE CAHILL, RUTH O'DONOVAN, JOSEF VAN GENABITH, and ANDY WAY. 2004. Evaluation of an automatic f-structure annotation algorithm against the PARC 700 dependency bank. In *Proceedings of the LFG04 Conference*, pages 101–121, Christchurch, New Zealand.

BÖHMOVÁ, ALENA, JAN HAJIČ, EVA HAJIČOVÁ, and BARBORA HLADKÁ. 2003. The Prague Dependency Treebank: A three level annotation scenario. In *Treebanks: building and using parsed corpora*, ed. by Anne Abeillé. Springer.

CAHILL, AOIFE, MICHAEL BURKE, RUTH O'DONOVAN, JOSEF VAN GENABITH, and ANDY WAY. 2004. Long-distance dependency resolution in automatically acquired wide-coverage PCFG-based LFG approximations. In *Proceedings of the 42nd Annual Meeting of the ACL*, pages 319–326, Barcelona, Spain.

CALLMEIER, ULRICH. 2000. PET - a platform for experimentation with efficient HPSG processing techniques. *Natural Language Engineering* 6(1):99–107.

CALLMEIER, ULRICH, 2001. Efficient parsing with large-scale unification grammars. Master's thesis, Saarland University.

CALLMEIER, ULRICH, ANDREAS EISELE, ULRICH SCHÄFER, and MELANIE SIEGEL. 2004. The deepthought core architecture framework. In *Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC 2004)*, pages Lisbon, Portugal, 1205–1208.

CARROLL, JOHN, TED BRISCOE, and ANTONIO SANFILIPPO. 1998. Parser evaluation: a survey and a new proposal. In *Proceedings of the 1st LREC Conference*, pages 447–454, Granada, Spain.

Carroll, John, Anette Frank, Dekang Lin, Detlef Prescher, and Hans Uszkoreit. 2002. Beyond PARSEVAL — towards improved evaluation measures for parsing systems. In *Proceedings of the Beyond PARSEVAL Workshop, Third International Conference on Language Resources and Evaluation (LREC 2002)*, pages 1–3, Granada, Spain.

Carroll, John, Guido Minnen, and Ted Briscoe. 1999. Corpus annotation for parser evaluation. In *Proceedings of the EACL Workshop on Linguistically Interpreted Corpora (LINC)*, Bergen, Norway.

Carroll, John, and Stephan Oepen. 2005. High efficiency realization for a wide-coverage grammar. In *Proceedings of the 2nd International Joint Conference on Natural Language Processing*, pages 165–176, Jeju Island, Korea.

Carter, David. 1997. The treebanker: a tool for supervised training of parsed corpora. In *Proceedings of a Workshop on Computational Environments for Grammar Development and Linguistic Engineering*, pages 9–15, Madrid, Spain.

Charniak, Eugene. 2000. A maximum-entropy-inspired parser. In *Proceedings of the First Conference of the North American chapter of the Association for Computational Linguistics*, pages 132–139, San Francisco, USA.

Charniak, Eugene, and Mark Johnson. 2005. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the 43rd Annual Meeting of the ACL*, pages 173–180, Ann Arbor, USA.

Clark, Stephen. 2002. Supertagging for combinatory categorical grammar. In *Proceedings of the 6th Interna-*

tional Workshop on Tree Adjoining Grammar and Related Frameworks, pages 101–106, Venice, Italy.

Clark, Stephen, and James R. Curran. 2004. The importance of supertagging for wide-coverage CCG parsing. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING 2004)*, pages 282–288, Geneva, Switzerland.

Clark, Stephen, and James R. Curran. 2007a. Formalism-independent parser evaluation with CCG and DepBank. In *Proceedings of the 45th Annual Meeting of the ACL*, pages 248–255, Prague, Czech Republic.

Clark, Stephen, and James R. Curran. 2007b. Wide-coverage efficient statistical parsing with CCG and log-linear models. *Computational Linguistics* 33(4):493–552.

Clark, Stephen, James R. Curran, and Miles Osbourne. 2003. Bootstrapping POS taggers using unlabelled data. In *Proceedings of the 7th Conference on Natural Language Learning (CoNLL-2003)*, pages 49–55, Edmonton, Canada.

Clark, Stephen, and Julia Hockenmaier. 2002. Evaluating a wide-coverage CCG parser. In *Proceedings of the Beyond PARSEVAL Workshop, Third International Conference on Language Resources and Evaluation (LREC 2002)*, pages 60–66, Granada, Spain.

Clark, Stephen, Julia Hockenmaier, and Mark Steedman. 2002. Building deep dependency structures with a wide-coverage CCG parser. In *Proceedings of the 40th Annual Meeting of the ACL and 3rd Annual Meeting of the NAACL (ACL-02)*, pages 327–334, Philadelphia, USA.

Collins, Michael. 1996. A new statistical parser based on bigram lexical dependencies. In *Proceedings of the 34th Annual Meeting of the ACL*, pages 184–191, San Francisco, USA.

Collins, Michael, 1999. *Head-Driven Statistical Models for Natural Language Parsing*. University of Pennsylvania dissertation.

Collins, Michael. 2000. Discriminative reranking for natural language parsing. In *Seventeenth International Conference on Machine Learning (ICML 2000)*, pages 175–182, Stanford, USA.

Collins, Michael, and Terry Koo. 2005. Discriminative reranking for natural language parsing. *Computational Linguistics* 31(1):25–70.

Copestake, Ann, Dan Flickinger, Ivan A. Sag, and Carl Pollard. 2005. Minimal recursion semantics: An introduction. *Research on Language and Computation* vol 3(no 4):pp 281–332.

Crouch, Richard, Ronald M. Kaplan, Tracy Holloway King, and Stefan Riezler. 2002. A comparison of evaluation metrics for a broad-coverage stochastic parser. In *Proceedings of the Beyond PARSEVAL Workshop, Third International Conference on Language Resources and Evaluation (LREC 2002)*, pages 67–74, Granada, Spain.

Crysmann, Berthold. 2003. On the efficient implementation of German verb placement in HPSG. In *Proceedings of RANLP 2003*, pages 112–116, Borovets, Bulgaria.

Crysmann, Berthold, Nuria Bertomeu, Peter Adolphs, Dan Flickinger, and Tina Kluwer. 2008.

Hybrid processing for grammar and style checking. In *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)*, pages 153–160, Manchester, UK.

Crysmann, Berthold, Anette Frank, Bernd Kiefer, Stefan Müller, Günter Neumann, Jakub Piskorski, Ulrich Schäfer, Melanie Siegel, Hans Uszkoreit, Feiyu Xu, Markus Becker, and Hans-Ulrich Krieger. 2002. An integrated architecture for shallow and deep processing. In *Proceedings of the 40th Annual Meeting of the ACL and 3rd Annual Meeting of the NAACL (ACL-02)*, pages 441–448, Philadelphia, USA.

Dalrymple, Mary. 2001. *Lexical Functional Grammar*. New York, USA: Academic Press.

Dalrymple, Mary. 2006. How much can part-of-speech tagging help parsing? *Natural Language Engineering* 12(4):373–389.

Daum, Michael, Kilian A. Foth, and Wolfgang Menzel. 2003. Constraint based integration of deep and shallow parsing techniques. In *Proceedings of EACL 2003*, Budapest.

Dickinson, Markus, and Charles Jochim. 2008. A simple method for tagset comparison. In *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC 2008)*, pages 821–826, Marrakech, Morocco.

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

Frank, Anette, Markus Becker, Berthold Crysmann, Bernd Kiefer, and Ulrich Schäfer. 2003. Integrated shallow and deep parsing: TopP meets HPSG. In *Proceedings of the 41st Annual Meeting of the ACL*, pages 104–111, Sapporo, Japan.

Gaizauskas, Robert. 1998. Evaluation in language and speech technology. *Computer Speech and Language* 12:249–262.

Gaizauskas, Robert, M. Hepple, and C. Huyck. 1998. A scheme for comparative evaluation of diverse parsing systems. In *Proceedings of the 1st LREC Conference*, pages 143–149, Granada, Spain.

Gazdar, Gerald, Ewan Klein, Geoffrey Pullum, and Ivan A. Sag. 1985. *Generalized phrase structure grammar*. Harvard University Press.

Goodman, Michael Wayne, and Francis Bond. 2009. Using generation for grammar analysis and error detection. In *Proceedings of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing*, pages 109–112, Suntec, Singapore.

Grishman, Ralph, Catherine Macleod, and John Sterling. 1992. Evaluation parsing strategies using standardized parse files. In *Proceedings of the Third Conference on Applied Natural Language Processing*, pages 156–161, Trento, Italy.

Grover, Claire, and Alex Lascarides. 2001. XML-based data preparation for robust deep parsing. In *Proceedings of the 39th Annual Meeting of the ACL and 10th Conference of the EACL (ACL-EACL 2001)*, pages 252–259, Toulouse, France.

Hara, Tadayoshi, Yusuke Miyao, and Jun'ichi Tsujii. 2007. Evaluating impact of re-training a lexical disambiguation model on domain adaptation of an HPSG parser. In *Proceedings of the 10th International Conference on Parsing Technology (IWPT 2007)*, pages 11–22, Prague, Czech Republic.

Hockenmaier, Julia, and Mark Steedman. 2007. CCGbank: A corpus of CCG derivations and dependency structures extracted from the Penn Treebank. *Computational Linguistics* 33(3):355–396.

Johnson, Mark, and Stefan Riezler. 2000. Exploiting auxiliary distributions in stochastic unification-based grammars. In *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*, pages 154–161, Seattle, USA.

Kaplan, Ronald M., and Joan Bresnan. 1982. Lexical-functional grammar: A formal system for grammatical representation. In *The Mental Representation of Grammatical Relations*, ed. by Joan Bresnan, pages 173–281. MIT Press.

Kaplan, Ronald M., Stefan Riezler, Tracy Holloway King, John T. Maxwell III, Alexander Vasserman, and Richard Crouch. 2004. Speed and accuracy in shallow and deep stochastic parsing. In *Proceedings of the 4th International Conference on Human Language Technology Research and 5th Annual Meeting of the NAACL (HLT-NAACL 2004)*, pages 97–104, Boston, USA.

Kiefer, Bernd, Hans-Ulrich Krieger, John Carroll, and Rob Malouf. 1999. A bag of useful techniques for efficient and robust parsing. In *Proceedings of the 37th Annual Meeting of the ACL*, pages 473–480, Maryland, USA.

Kim, Jong-Bok, and Jaehyung Yang. 2004. Projections from morphology to syntax in the korean resource grammar: Implementing typed feature structures. In *Proceedings of the 5th International Conference of Computational Linguistics and Intelligent Text Processing (CICLing 2004)*, pages 14–25, Seoul, Korea.

King, Tracy Holloway, Richard Crouch, Stefan Riezler, Mary Dalrymple, and Ronald M. Kaplan. 2003. The PARC 700 Dependency Bank. In *Proceedings of the LINC-03 Workshop*, Budapest, Hungary.

Kordoni, Valia, and Julia Neu. 2004. Deep analysis of modern greek. In *Proceedings of the First International Joint Conference on Natural Language Processing (IJCNLP-04)*, pages 674—-683, Hainan Island, China.

Leech, Geoffrey. 1997. Grammatical tagging. In *Corpus Annotation: Linguistic Information from Computer Text Corpora*, ed. by Roger Garside, Geoffrey Leech, and Anthong McEnery, chapter 2. Addison Wesley Longman Ltd.

Lin, Dekang. 1998. A dependency-based method for evaluating broad-coverage parsers. *Natural Language Engineering* 4(2):97–114.

Magerman, David M., 1994. *Natural Language Parsing as Statistical Pattern Recognition*. Stanford University dissertation.

Marcus, Mitchell, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. 1994. The Penn Treebank: Annotating predicate argument structure. In *Proceedings of the Human Language Technology Workshop*, pages 114–119, Princeton, USA.

Marcus, Mitchell P., Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics* 19(2):313–330.

Marimon, Montserrat, Núria Bel, and Natalia Seghezzi. 2007. Test-suite construction for a Spanish grammar. In *Proceedings of the GEAF 2007 Workshop*, pages 224–237, Stanford, USA.

Matsumoto, Yuuji, Akira Kitauchi, Tatsu Yamashita, and Yoshitake Hirano. 1999. Japanese morphological analysis system ChaSen version 2.0 manual. Technical Report NAIST-IS-TR99009, NAIST, Nara, Japan.

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

Maxwell, III, John T., and Ronald M. Kaplan. 1993. The interface between phrasal and functional constraints. *Computational Linguistics* 19(4):571–589.

McClosky, David, Eugene Charniak, and Mark Johnson. 2006a. Effective self-training for parsing. In *Proceedings of the Human Language Technology Conference of the NAACL*, pages 152–159, New York City, USA.

McClosky, David, Eugene Charniak, and Mark Johnson. 2006b. Reranking and self-training for parser adaptation. In *Proceedings of the 44th Annual Meeting of the ACL and the 21st International Conference on Computational Linguistics*, pages 337–344, Sydney, Australia.

McDonald, Ryan, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP)*, pages 523–530, Vancouver, Canada.

Miyao, Yusuke, Takashi Ninomiya, and Jun'ichi Tsujii. 2004. Corpus-oriented grammar development for acquiring a Head-driven Phrase Structure Grammar from the Penn Treebank. In *Proceedings of the First International Joint Conference on Natural Language Processing (IJCNLP-04)*, Hainan Island, China.

Miyao, Yusuke, Rune Sætre, Kenji Sagae, Takuya Matsuzaki, and Jun'ichi Tsujii. 2008. Task-oriented evaluation of syntactic parsers and their representations. In *Proceedings of the 46th Annual Meeting of the ACL*, pages 46–54, Columbus, USA.

Miyao, Yusuke, Kenji Sagae, and Jun'ichi Tsujii. 2007. Towards framework-independent evaluation of deep linguistic parsers. In *Proceedings of the GEAF 2007 Workshop*, Palo Alto, California.

Miyao, Yusuke, and Jun'ichi Tsujii. 2005. Probabilistic disambiguation models for wide-coverage HPSG parsing. In *Proceedings of the 43rd Annual Meeting of the ACL*, pages 83–90, Ann Arbor, USA.

Miyao, Yusuke, and Jun'ichi Tsujii. 2008. Feature forest models for probabilistic HPSG parsing. *Computational Linguistics* 34(1):35–80.

Müller, Stefan, and Walter Kasper. 2000. HPSG analysis of German. In *Verbmobil: Foundations of Speech-*

to-Speech Translation, pages 238–253. Berlin, Germany: Springer.

NINOMIYA, TAKASHI, TAKUYA MATSUZAKI, YOSHIMASA TSURUOKA, YUSUKE MIYAO, and JUN'ICHI TSUJII. 2006. Extremely lexicalised models for accurate and fast HPSG parsing. In *Proceedings of the 2006 Conference on Empirical Methods in natural Language Processing (EMNLP 2006)*, pages 155–163, Sydney, Australia.

NIVRE, JOAKIM, JOHAN HALL, and JENS NILSSON. 2004. Memory-based dependency parsing. In *Proceedings of the 8th Conference on Natural Language Learning (CoNLL-2004)*, pages 49–56, Boston, USA.

OEPEN, STEPHAN, and JOHN CARROLL. 2000. Ambiguity packing in constraint-based parsing - practical results. In *Proceedings of the 1st Conference of the North American Chapter of the Association for Computational Linguistics*, pages 162–169, Seattle, USA.

OEPEN, STEPHAN, 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. 2004a. Somå 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.

OEPEN, STEPHAN, DAN FLICKINGER, KRISTINA TOUTANOVA, and CHRISTOPHER D. MANNING. 2004b. LinGO redwoods. a rich and dynamic treebank for HPSG.

Journal of Research in Language and Computation 2(4):575–596.

Oepen, Stephan, and Jan Tore Lønning. 2006. Discriminant-based MRS banking. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC 2006)*, pages 1250–1255, Genoa, Italy.

Plank, Barbara, and Gertjan van Noord. 2008. Exploring an auxiliary distribution based approach to domain adaptation of a syntactic disambiguation model. In *Proceedings of the workshop on Cross-Framework and Cross-Domain Parser Evaluation*, pages 9–16, Manchester, UK.

Pollard, Carl, and Ivan A. Sag. 1994. *Head-Driven Phrase Structure Grammar*. Chicago, USA: University of Chicago Press.

Prins, Robbert, and Gertjan van Noord. 2003. Reinforcing parser preferences through tagging. *Traitement Automatique des Langues* 44(3):121–139.

Riezler, Stefan, Tracy Holloway King, Richard S. Crouch, John T Maxwell, and Ronald M. Kaplan. 2002. Parsing the Wall Street Journal using a lexical-functional grammar and discriminative estimation techniques. In *Proceedings of the 40th Annual Meeting of the ACL and 3rd Annual Meeting of the NAACL (ACL-02)*, pages 7–12, Philadelphia, USA.

Rimell, Laura, and Stephen Clark. 2008a. Adapting a lexicalized-grammar parser to contrasting domains. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing (EMNLP 2008)*, pages 475–484, Honolulu, USA.

Rimell, Laura, and Stephen Clark. 2008b. Constructing a parser evaluation scheme. In *Coling 2008: Proceedings of the workshop on Cross-Framework and Cross-Domain Parser Evaluation*, pages 44–50, Manchester, UK.

Roark, Brian. 2002. Evaluating parser accuracy using edit distance. In *Proceedings of the Beyond PARSEVAL Workshop, Third International Conference on Language Resources and Evaluation (LREC 2002)*, pages 30–36, Granada, Spain.

Sagae, Kenji, Yusuke Miyao, and Jun'ichi Tsujii. 2007. HPSG parsing with shallow dependency constraints. In *Proceedings of the 45th Annual Meeting of the ACL*, pages 624–631, Prague, Czech Republic.

Sampson, Geoffrey, and Anna Babarczy. 2002. A test of the leaf-ancestor metric for parse accuracy. In *Proceedings of the Beyond PARSEVAL Workshop, Third International Conference on Language Resources and Evaluation (LREC 2002)*, pages 23–29, Granada, Spain.

Schabes, Yves, and Aravind K. Joshi. 1991. Parsing with lexicalized tree adjoining grammar. In *Current Issues in Parsing Technology*, ed. by Masaru Tomita, chapter 3, pages 25–48. Kluwer.

Schäfer, Ulrich. 2006. Middleware for creating and combining multi-dimensional NLP markup. In *Proceedings of the EACL-2006 Workshop on Multi-dimensional Markup in Natural Language Processing*, pages 81–84, Trento, Italy.

Sekine, Satoshi, and Michael Collins, 1997. EvalB: a bracket scoring program.

Siegel, Melanie, and Emily M. Bender. 2002. 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*, pages 31–38, Taipei, Taiwan.

Steedman, Mark. 2000. *The Syntactic Process*. MIT Press.

Torisawa, Kentaro, Kenji Nishida, Yusuke Miyao, and Jun'ichi Tsujii. 2000. An HPSG parser with CFG filtering. *Natural Language Engineering* 6(1):pp 63–80.

Toutanova, Kristina, Chistopher D. Manning, Stuart M. Shieber, Dan Flickinger, and Stephan Oepen. 2002. Parse disambiguation for a rich HPSG grammar. In *First Workshop on Treebanks and Linguistic Theories (TLT2002)*, pages 253–263.

Tseng, Jesse. 2003. Lkb grammar development: French and beyond. In *Workshop on Ideas and Strategies for Multilingual Grammar Development*, pages 91–97, Vienna, Austria.

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

van Noord, Gertjan, and Robert Malouf. 2004. Wide coverage parsing with stochastic attribute value grammars. In *IJCNLP-04 Workshop Beyond Shallow Analyses – Formalisms and statistical modelling for deep analyses*.

Yamada, Hiroyasu, and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of the 8th International Workshop on Parsing Technologies*, pages 195–206, Nancy, France.

Ytrestøl, Gisle, Dan Flickinger, and Stephan Oepen. 2009. Extracting and annotating wikipedia sub-domains. In *Proceedings of the Seventh International Workshop on Treebanks and Linguistic Theories (TLT 7)*, pages 185–196, Groningen, The Netherlands.

Zhang, Yi, Timothy Baldwin, and Valia Kordoni. 2007a. The corpus and the lexicon: Standardising deep lexical acquisition evaluation. In *Proceedings of the ACL 2007 Workshop on Deep Linguistic Processing*, pages 152–159, Prague, Czech Republic.

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

Zhang, Yi, Stephan Oepen, and John Carroll. 2007b. Efficiency in unification-based n-best parsing. In *Proceedings of the 10th international conference on parsing technologies (IWPT 2007)*, pages 48–59, Prague, Czech Republic.

A Generic Lexical Types

Generic lexical types triggered by the default unknown word handling mechanism occur in some of the training data for the supertaggers. Where possible, these generic types were mapped to an appropriate native lexical type. The list below gives each of these types, and the mapping where appropriate.

Generic Lexical Type	Mapped to
aj_-_i-unk_le	aj_-_i_le
aj_-_i-cmp-unk_le	n/a
aj_-_i-sup-unk_le	n/a
aj_-_i-crd-unk_le	n/a
av_-_i-unk_le	n/a
n_-_c-pl-unk_le	n_-_c-pl_le
n_-_mc-unk_le	n_-_mc_le
n_-_pn-unk_le	n_-_pn_le
v_np*_bse-unk_le	n/a
v_np*_pr-3s-unk_le	n/a
v_np*_pr-n3s-unk_le	n/a
v_np*_pa-unk_le	n/a
v_np*_prp-unk_le	n/a
v_np*_psp-unk_le	n/a
v_-_pas-unk_le	n/a