Ubertagging: Joint segmentation and supertagging for English

Rebecca Dridan
Institutt for Informatikk
Universitetet i Oslo
rdridan@ifi.uio.no

Abstract

A precise syntacto-semantic analysis of English requires a large detailed lexicon with the possibility of treating multiple tokens as a single meaning-bearing unit, a *word-with-spaces*. However parsing with such a lexicon, as included in the English Resource Grammar, can be very slow. We show that we can apply supertagging techniques over an ambiguous token lattice without resorting to previously used heuristics, a process we call ubertagging. Our model achieves an ubertagging accuracy that can lead to a four to eight fold speed up while improving parser accuracy.

1 Introduction and Motivation

Over the last decade or so, supertagging has become a standard method for increasing parser efficiency for heavily lexicalised grammar formalisms such as LTAG (Bangalore and Joshi, 1999), CCG (Clark and Curran, 2007) and HPSG (Matsuzaki et al., 2007). In each of these systems, fine-grained lexical categories, known as supertags, are used to prune the parser search space prior to full syntactic parsing, leading to faster parsing at the risk of removing necessary lexical items. Various methods are used to configure the degree of pruning in order to balance this trade-off.

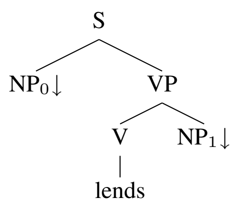
The English Resource Grammar (ERG; Flickinger (2000)) is a large hand-written HPSG-based grammar of English that produces fine-grained syntacto-semantic analyses. Given the high level of lexical ambiguity in its lexicon, parsing with the ERG should therefore also benefit from supertagging, but while various attempts have shown possibilities (Blunsom, 2007; Dridan et al., 2008; Dridan, 2009), supertagging is still not a standard element in the ERG parsing pipeline. There are two main reasons for this. The first is that the ERG lexicon does not assign simple atomic categories to words, but instead builds complex structured *signs* from information about lemmas and lexical rules, and hence the shape and integration of the supertags is not straightforward. Bangalore and Joshi (2010) define a supertag as a primitive structure that contains all the information about a lexical item, including argument structure, and where the arguments should be found. Within the ERG, that information is not all contained in the lexicon, but comes from different places. The choice, therefore, of what information may be predicted prior to parsing and how it should be integrated into parsing is an open question.

The second reason that supertagging is not standard with ERG processing is one that is rarely considered when processing English, namely ambiguous segmentation. In most mainstream English parsing, the segmentation of parser input into tokens that will become the leaves of the parse tree is considered a fixed, unambiguous process. While recent work (Dridan and Oepen, 2012) has shown that producing even these tokens is not a solved problem, the issue we focus on here is the ambiguous mapping from these tokens to meaning-bearing units that we might call *words*. Within the ERG lexicon are many multi-token lexical entries that are sometimes referred to as *words-with-spaces*. These multi-token entries are added to the lexicon where the grammarian finds that the semantics of a fixed expression is non-compositional and has the distributional properties of other single word entries. Some examples include an adverb-like *all of a sudden*, a preposition-like *for example* and an adjective-like *over and done with*. Each of these entries create an segmentation ambiguity between treating the whole expression as a single unit, or allowing analyses comprising entries triggered by the individual tokens. Previous supertagging research using the ERG has either used the gold standard tokenisation, hence making the task artificially easier, or else tagged the individual tokens, using various heuristics to apply multi-token tags to single tokens. Neither approach has been wholly satisfactory.

In this work we avoid the heuristic approaches and learn a sequential classification model that can simultaneously determine the most likely segmentation and supertag sequences, a process we dub ubertagging. We also experiment with more fine-grained tag sets than have been previously used, and find that it is possible to achieve a level of ubertagging accuracy that can improve both parser speed and accuracy for a precise semantic parser.

2 Previous Work

As stated above, supertagging has become a standard tool for particular parsing paradigms, but the definitions of a supertag, the methods used to learn them, and the way they are used in parsing varies across formalisms. The original supertags were 300 LTAG elementary trees, predicted using a fairly simple trigram tagger that provided a configurable number of tags per token, since the tagger was not accurate enough to make assigning a single tree viable parser input (Bangalore and Joshi, 1999). The C&C CCG parser uses a more complex Maximum Entropy tagger to assign tags from a set of 425 CCG lexical categories (Clark and Curran, 2007). They also found it necessary to supply more than one tag per token, and hence assign all tags that have a probability within a percentage β of the most likely tag for each token. Their standard parser configuration uses a very restrictive β value initially, relaxing it when no parse can be found. Matsuzaki et al. (2007) use a supertagger similar to the C&C tagger alongside a CFG filter to improve the speed of their HPSG parser, feeding sequences of single tags to the parser until a parse is possible. As in the ERG, category and inflectional information are separate in the automatically-extracted ENJU grammar: their supertag set consists of 1361 tags constructed by combining lexical categories and lexical rules. Figure 1 shows examples of supertags from these three tag sets, all describing the simple transitive use of *lends*.



(a) LTAG

(S[dcl]\NP)/NP

(b) CCG

[NP.nom<V.bse>NP.acc]-singular3rd_verb_rule

(c) ENJU HPSG

Figure 1: Examples of supertags from LTAG, CCG and ENJU HPSG, for the word *lends*.

The ALPINO system for parsing Dutch is the closest in spirit to our ERG parsing setup, since it also uses a hand-written HPSG-based grammar, including multi-token entries in its lexicon. Prins and van Noord (2003) use a trigram HMM tagger to calculate the likelihood of up to 2392 supertags, and discard those that are not within τ of the most likely tag. For their multi-token entries, they assign a constructed category to each token, so that instead of assigning `preposition` to the expression *met betrekking tot* ("with respect to"), they use `(1,preposition)`, `(2,preposition)`, `(3,preposition)`. Without these constructed categories, they would only have 1365 supertags.

Most previous supertagging attempts with the ERG have used the grammar's lexical types, which describe the coarse-grained part of speech, and the subcategorisation of a word, but not the inflection. Hence both *lends* and *lent* have a possible lexical type `v_np*_pp*_to_le`, which indicates a verb, with optional noun phrase and prepositional phrase arguments, where the preposition has the form *to*. The number of lexical types changes as the grammar grows, and is currently just over 1000. Dridan (2009) and Fares (2013) experimented with other tag types, but both found lexical types to be the optimal balance between predictability and efficiency. Both used a multi-tagging approach dubbed selective tagging to integrate the supertags into the parser. This involved only applying the supertag filter when the tag probability is above a configurable threshold, and not pruning otherwise.

For multi-token entries, both Blunsom (2007) and

adverb	adverb	adverb
adverb	ditto	ditto
1,adverb	2,adverb	3,adverb
all	*in*	*all*

Figure 2: Options for tagging parts of the multi-token adverb *all in all* separately.

Dridan (2009) assigned separate tags to each token, with Blunsom (2007) assigning a special `ditto` tag all but the initial token of a multi-token entry, while Dridan (2009) just assigned the same tag to each token (leading to *example* in the expression *for example* receiving `p_np_i_le`, a preposition-type category). Both of these solutions (demonstrated in Figure 2), as well as that of Prins and van Noord (2003), in some ways defeat one of the purposes of treating these expressions as fixed units. The grammarian, by assigning the same category to, for example, *all of a sudden* and *suddenly*, is declaring that these two expressions have the same distributional properties, the properties that a sequential classifier is trying to exploit. Separating the tokens loses that information, and introduces extra noise into the sequence model.

Ytrestøl (2012) and Fares (2013) treat the multi-entry tokens as single expressions for tagging, but with no ambiguity. Ytrestøl (2012) manages this by using gold standard tokenisation, which is, as he states, the standard practice for statistical parsing, but is an artificially simplified setup. Fares (2013) is the only work we know about that has tried to predict the final segmentation that the ERG produces. We compare segmentation accuracy between our joint model and his stand-alone tokeniser in Section 6.

Looking at other instances of joint segmentation and tagging leads to work in non-whitespace separated languages such as Chinese (Zhang and Clark, 2010) and Japanese (Kudo et al., 2004). While at a high level, this work is solving the same problem, the shape of the problems are quite different from a data point of view. Regular joint morphological analysis and segmentation has much greater ambiguity in terms of possible segmentations but, in most cases, less ambiguity in terms of labelling than our situation. This also holds for other lemmatisation and morphological research, such as Toutanova and Cherry (2009). While we drew inspiration from this related area, as well as from the speech recognition field, differences in the relative frequency of observations and labels, as well as in segmentation ambiguity mean that conclusions found in these areas did not always hold true in our problem space.

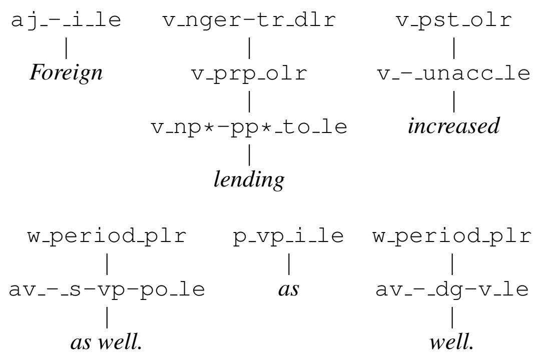


Figure 3: A selection from the 70 **lexitems** instantiated for *Foreign lending increased as well.*

3 The Parser

The parsing environment we work with is the PET parser (Callmeier, 2000), a unification-based chart parser that has been engineered for efficiency with precision grammars, and incorporates subsumption-based ambiguity packing (Oepen and Carroll, 2000) and statistical model driven selective unpacking (Zhang et al., 2007). Parsing in PET is divided in two stages. The first stage, lexical parsing, covers everything from tokenising the raw input string to populating the base of the parse chart with the appropriate lexical items, ready for the second — syntactic parsing — stage. In this work, we embed our ubertagging model between the two stages. By this point, the input has been segmented into what we call `internal tokens`, which broadly means splitting at whitespace and hyphens, and making *'s* a separate token. These tokens are subject to a morphological analysis component which proposes possible inflectional and derivational rules based on word form, and then are used in retrieving possible lexical entries from the lexicon. The results of applying the appropriate lexical rules, plus affixation rules triggered by punctuation, to the lexical entries form a lexical item object, that for this work we dub a **lexitem**.

Figure 3 shows some examples of **lexitems** instantiated after the lexical parsing stage when analysing *Foreign lending increased as well.* The pre-terminal labels on these subtrees are the lexical types that have previously been used as supertags for the ERG. For uninflected words, with no punctuation affixed, the lexical type is the only element in the **lexitem**, other than the word form (e.g. *Foreign*, *as*). In this example, we also see **lexitems** with inflectional rules (`v_prp_olr`, `v_pst_olr`), derivational rules (`v_nger-tr_dlr`) and punctuation affixation rules (`w_period_plr`).

These **lexitems** are put in to a chart, forming a lexical lattice, and it is over this lattice that we apply our ubertagging model, removing unlikely **lexitems** before they are seen by the syntactic parsing stage.

4 The Data

The primary data sets we use in these experiments are from the 1.0 version of DeepBank (Flickinger et al., 2012), an HPSG annotation of the Wall Street Journal text used for the Penn Treebank (PTB; Marcus et al. (1993)). The current version has gold standard annotations for approximately 85% of the first 22 sections. We follow the recommendations of the DeepBank developers in using Sections 00–19 for training, Section 20 (WSJ_{20}) for development and Section 21 (WSJ_{21}) as test data.

In addition, we use two further sources of training data: the training portions of the LinGO Redwoods Treebank (Oepen et al., 2004), a steadily growing collection of gold standard HPSG annotations in a variety of domains; and the Wall Street Journal section of the North American News Corpus (NANC), which has been parsed, but *not* manually annotated. This builds on observations by Prins and van Noord (2003), Dridan (2009) and Ytrestøl (2012) that even uncorrected parser output makes very good training data for a supertagger, since the constraints in the parser lead to viable, if not entirely correct sequences. This allows us to use much larger training sets than would be possible if we required manually annotated data.

In final testing, we also include two further data sets to observe how domain affects the contribution of the ubertagging. These are both from the test portion of the Redwoods Treebank: CatB, an essay about open-source software;[1] and WeScience_{13}, text from Wikipedia articles about Natural Language Processing from the WeScience project (Ytrestøl et al., 2009). Table 1 summarises the vital statistics of the data we use.

With the focus on multi-token **lexitems**, it is instructive to see just how frequent they are. In terms of **type** frequency, almost 10% of the approximately 38500 lexical entries in the current ERG lexicon have more than one token in their canonical form.[2] However, while this is a significant percentage of the lexicon, they do not account for the same percentage of **tokens** during parsing. An analysis of $\text{WSJ}_{00:19}$ shows that approximately one third of the sentences had at least one multi-token **lexitem** in the unpruned lexical lattice, and in just under half of those, the gold standard analysis included a multi-word entry. That gives the multi-token **lexitems** the awkward property of being rare enough to be difficult for a statistical classifier to accurately detect (just under 1% of the leaves of gold parse trees contain multiple tokens), but too frequent to ignore. In addition, since these multi-token expressions have often been distinguished because they are non-compositional, failing to detect the multi-word usage can lead to a disproportionately adverse effect on the semantic analysis of the text.

5 Ubertagging Model

Our ubertagging model is very similar to a standard trigram Hidden Markov Model (HMM), except that the states are not all of the same length. Our states are based on the **lexitems** in the lexical lattice produced by the lexical parsing stage of PET, and as such, can be partially overlapping. We formalise this be defining each state by its start position, end position, and tag. This turns out to make our model equivalent to a type of Hidden semi-Markov Model called a segmental HMM in Murphy (2002). In a segmental HMM, the states are segments with a tag (t) and a length in frames (l). In our setup, the frames are the ERG internal tokens and the segments are the **lexitems**, which are the potential candidates

[1]`http://catb.org/esr/writings/cathedral-bazaar/` by Eric S. Raymond

[2]While the parser has mechanisms for handling words unknown to the lexicon, with the current grammar these mechanisms will never propose a multi-token **lexitem**, and so only the multi-token entries explicitly in the lexicon will be recognised as such.

Data Set	Source	Use	Gold?	Trees	Lexitems All	Lexitems M-T
WSJ$_{00:19}$	DeepBank 1.0 §00–19	train	yes	33783	661451	6309
Redwoods	Redwoods Treebank	train	yes	39478	432873	6568
NANC	LDC2008T15	train	no	2185323	42376523	399936
WSJ$_{20}$	DeepBank 1.0 §20	dev	yes	1721	34063	312
WSJ$_{21}$	DeepBank 1.0 §21	test	yes	1414	27515	253
WeScience$_{13}$	Redwoods Treebank	test	yes	802	11844	153
CatB	Redwoods Treebank	test	yes	608	11653	115

Table 1: Test, development and training data used in these experiments. The final two columns show the total number of **lexitems** used for training (**All**), as well as how many of those were multi-token **lexitems** (**M-T**).

to become leaves of the parse tree. As indicated above, the majority of segments (over 99%) will be one frame long, but segments of up to four frames are regularly seen in the training data.

A standard trigram HMM has a transition probability matrix A, where the elements A_{ijk} represent the probability $P(k|ij)$, and an emission probability matrix B whose elements B_{jo} record the probabilities $P(o|j)$. Given these matrices and a vector of observed frames, O, the posterior probabilities of each state at frame v are calculated as:[3]

$$P(q_v = q_y|O) = \frac{\alpha_v(q_y)\beta_v(q_y)}{P(O)} \tag{1}$$

where $\alpha_v(q_y)$ is the forward probability at frame v, given a current state q_y (i.e. the probability of the observation up to v, given the state):

$$\alpha_v(q_y) \equiv P(O_{0:v}|q_v = q_y) \tag{2}$$

$$= \sum_{q_x} \alpha_v(q_x q_y) \tag{3}$$

$$\alpha_v(q_x q_y) = B_{q_y o_v} \sum_{q_w} \alpha_{v-1}(q_w q_x) A_{q_w q_x q_y} \tag{4}$$

$\beta_v(q_y)$ is the backwards probability at frame v, given a current state q_y (the probability of the observation from v, given the state):

$$\beta_v(q_y) \equiv P(O_{v+1:V}|q_v = q_y) \tag{5}$$

$$= \sum_{q_x} \beta_v(q_x q_y) \tag{6}$$

$$\beta_v(q_x q_y) = \sum_{q_z} \beta_{v+1}(q_y q_z) A_{q_x q_y q_z} B_{q_z o_{v+1}} \tag{7}$$

and the probability of the full observation sequence is equal to the forward probability at the end of the sequence, or the backwards probability at the start of the sequence:

$$P(O) = \alpha_V(\langle\mathcal{E}\rangle) = \beta_0(\langle\mathcal{S}\rangle) \tag{8}$$

In implementation, our model varies only in what we consider the previous or next states. While v still indexes frames, q_v now indicates a state that *ends* with frame v, and we look forwards and backwards to adjacent states, not frames, formally designated in terms of l, the length of the state. Hence, we modify equation (4):

$$\alpha_v(q_x q_y) = B_{q_y O_{v-l+1:v}} \sum_{q_w} \alpha_{v-l}(q_w q_x) A_{q_w q_x q_y} \tag{9}$$

where $v-l$ indexes the frame before the current state *starts*, and hence we are summing over all states that lead directly to our current state. An equivalent modification to equation (7) gives:

$$\beta_v(q_x q_y) = \sum_{\substack{q_z \\ \in Q_n}} \sum_{l(q_z)} \beta_{v+l}(q_y q_z) A_{q_x q_y q_z} B_{q_z O_{v+1:v+l}} \tag{10}$$

[3]Since we will require per-state probabilities for integration to the parser, we focus on the calculation of posterior probabilities, rather than determing the single best path.

Type	Example	#Tags
LTYPE	v_np-pp*_to_le	1028
INFL	v_np-pp*_to_le:v_pas_odlr	3626
FULL	v_np-pp*_to_le:v_pas_odlr:w_period_plr	21866

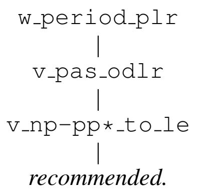


Figure 4: Possible tag types and their tag set size, with examples derived from the **lexitem** on the right.

where Q_n is the set of states that start at $v+1$ (i.e., the states immediately following the current state), and $l(q_z)$ is the length of state q_z.

We construct the transition and emission probability matrices using relative frequencies directly observed from the training data, where we make the simplifying assumption that $P(q_k|q_iq_j) \equiv P(t(q_k)|t(q_i)t(q_k))$. Which is to say, while **lexitems** with the same tag, but different length will trigger distinct states with distinct emission probabilities, they will have the same transition probabilities, given the same proceeding tag.[4] Even with our large training set, some tag trigrams are rare or unseen. To smooth these probabilities, we use deleted interpolation to calculate a weighted sum of the trigram, bigram and unigram probabilities, since it has been successfully used in effective PoS taggers like the TnT tagger (Brants, 2000). Future work will look more closely at the effects of different smoothing methods.

6 Intrinsic Ubertag Evaluation

In order to develop and tune the ubertagging model, we first looked at segmentation and tagging performance in isolation over the development set. We looked at three tag granularities: lexical types (LTYPE) which have previously been shown to be the optimal granularity for supertagging with the ERG, inflected types (INFL) which encompass inflectional and derivational rules applied to the lexical type, and the full lexical item (FULL), which also includes affixation rules used for punctuation handling. Examples of each tag type are shown in Figure 4, along with the number of tags of each type seen in the training data.

[4]Since the multi-token lexical entries are defined because they have the same properties as the single token variants, there is no reason to think the length of a state should influence the tag sequence probability.

	Segmentation		Tagging	
Tag Type	**F1**	**Sent.**	**F1**	**Sent.**
FULL	99.55	94.48	93.92	42.13
INFL	99.45	93.55	93.74	41.49
LTYPE	99.40	93.03	93.27	38.12

Table 2: Segmentation and tagging performance of the best path found for each model, measured per segment in terms of F_1, and also as complete sentence accuracy.

Single sequence results Table 2 shows the results when considering the best path through the lattice. In terms of segmentation, our sentence accuracy is comparable to that of the stand-alone segmentation performance reported by Fares et al. (2013) over similar data.[5] In that work, the authors used a binary CRF classifier to label points between objects they called micro-tokens as either SPLIT or NOSPLIT. The CRF classifier used a less informed input (since it was external to the parser), but a much more complex model, to produce a best single path sentence accuracy of 94.06%. Encouragingly, this level of segmentation performance was shown in later work to produce a viable parser input (Fares, 2013).

Switching to the tagging results, we see that the F_1 numbers are quite good for tag sets of this size.[6] The best tag accuracy seen for ERG LTYPE-style tags was 95.55 in Ytrestøl (2012), using gold standard segmentation on a different data set. Dridan (2009) experimented with a tag granularity similar to our INFL (letype+morph) and saw a tag accuracy of 91.51, but with much less training data. From other formalisms, Kummerfeld et al. (2010)

[5]Fares et al. (2013) used a different section of an earlier version of DeepBank, but with the same style of annotation.

[6]We need to measure F_1 rather than tag accuracy here, since the number of tokens tagged will vary according to the segmentation.

report a single tag accuracy of 95.91, with the smaller CCG supertag set. Despite the promising tag F_1 numbers however, the sentence level accuracy still indicates a performance level unacceptable for parser input. Comparing between tag types, we see that, possibly surprisingly, the more fine-grained tags are more accurately assigned, although the differences are small. While instinctively a larger tag set should present a more difficult problem, we find that this is mitigated both by the sparse lexical lattice provided by the parser, and by the extra constraints provided by the more informative tags.

Multi-tagging results The multi-tagging methods from previous supertagging work becomes more complicated when dealing with ambiguous tokenisation. Where, in other setups, one can compare tag probabilities for all tags for a particular token, that no longer holds directly when tokens can partially overlap. Since ultimately, the parser uses **lexitems** which encompass segmentation and tagging information, we decided to use a simple integration method, where we *remove* any **lexitem** which our model assigns a probability below a certain threshold (ρ). The effect of the different tag granularities is now mediated by the relationship between the states in the ubertagging lattice and the **lexitems** in the parser's lattice: for the FULL model, this is a one-to-one relationship, but states from the models that use coarser-grained tags may affect multiple **lexitems**. To illustrate this point, Figure 5 shows some **lexitems** for the token *forecast,*, where there are multiple possible analyses for the comma. A FULL tag of `v_cp_le:v_pst_olr:w_comma_plr` will select only **lexitem** (b), whereas an INFL tag `v_cp_le:v_pst_olr` will select (b) and (c) and the LTYPE tag `v_cp_le` picks out (a), (b) and (c). On the other hand, where there is no ambiguity in inflection or affixation, an LTYPE tag of `n_-_mc_le` may relate to only a single **lexitem** ((f) in this case).

Since we are using an absolute, rather than relative, threshold, the number needs to be tuned for each model[7] and comparisons between models can only be made based on the effects (accuracy or pruning power) of the threshold. Table 3 shows how a selection of threshold values affect the accuracy

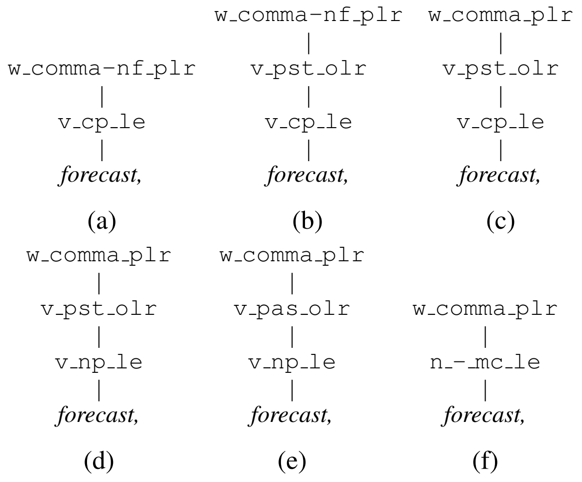


Figure 5: Some of the **lexitems** triggered by *forecast,* in *Despite the gloomy forecast, profits were up.*

Tag Type	ρ	Acc.	Lexitems Kept	Lexitems Ave.
FULL	0.00001	99.71	41.6	3.34
FULL	0.0001	99.44	33.1	2.66
FULL	0.001	98.92	25.5	2.05
FULL	0.01	97.75	19.4	1.56
INFL	0.0001	99.67	37.9	3.04
INFL	0.001	99.25	29.0	2.33
INFL	0.01	98.21	21.6	1.73
INFL	0.02	97.68	19.7	1.58
LTYPE	0.0002	99.75	66.3	5.33
LTYPE	0.002	99.43	55.0	4.42
LTYPE	0.02	98.41	43.5	3.50
LTYPE	0.05	97.54	39.4	3.17

Table 3: Accuracy and ambiguity after pruning **lexitems** in WSJ_{20}, at a selection of thresholds ρ for each model. Accuracy is measured as the percentage of gold **lexitems** remaining after pruning, while ambiguity is presented both as a percentage of **lexitems** kept, and the average number of **lexitems** per initial token still remaining.

[7]A tag set size of 1028 will lead to higher probabilities in general than a tag set size of 21866.

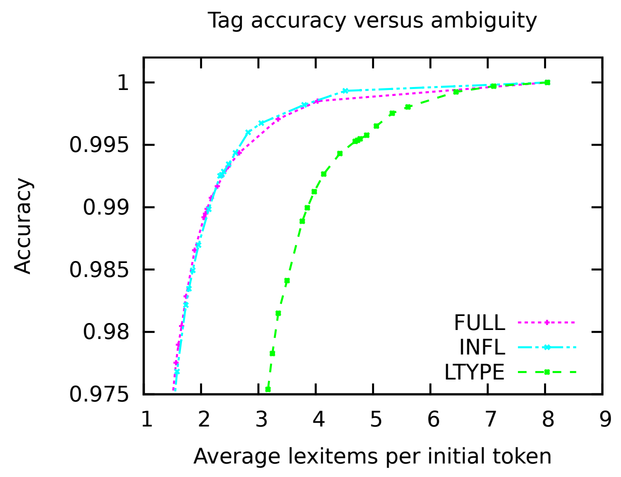


Figure 6: Accuracy over gold **lexitems** versus average **lexitems** per initial token over the development set, for each of the different ubertagging models.

and pruning impact of our different disambiguation models, where the accuracy is measured in terms of percentage of gold **lexitems** retained. The pruning effect is given both as percentage of **lexitems** retained after pruning, and average number of **lexitems** per initial token.[8] Comparison between the different models can be more easily made by examining Figure 6. Here we see clearly that the LTYPE model provides much less pruning for any given level of **lexitem** accuracy, while the performance of the other models is almost indistinguishable.

Analysis The current state-of-the-art POS tagging accuracy (using the 45 tags in the PTB) is approximately 97.5%. The most restrictive ρ value we report for each model was selected to demonstrate that level of accuracy, which we can see would lead to pruning over 80% of **lexitems** when using FULL tags, an average of 1.56 tags per token. While this level of accuracy has been sufficient for statistical treebank parsing, previous work (Dridan, 2009) has shown that tag accuracy cannot directly predict parser performance, since errors of different types can have very different effects. This is hard to quantify without parsing, but we made a qualitative analysis at the **lexitems** that were incorrectly being pruned. For all models, the most difficult **lexitems** to get correct were proper nouns, particular those that are also used as common nouns (e.g. *Bank*, *Airline*, *Report*). While capitalisation provides a clue here, it is not always deterministic, particularly since the treebank incorporates detailed decisions regarding the distinction between a name and a capitalised common noun that require real world knowledge, and are not necessarily always consistent. Almost two thirds of the errors made by the FULL and INFL models are related to these decisions, but only about 40% for the LTYPE model. The other errors are predominately over noun and verb type **lexitems**, as the open classes, with the only difference between models being that the FULL model seems marginally better at classifying verbs. The next section describes the end-to-end setup and results when parsing the development set.

[8]The average number of **lexitems** per token for the unrestricted parser is 8.03, although the actual assignment is far from uniform, with up to 70 **lexitems** per token seen for the very ambiguous tokens.

7 Parsing

With encouraging ubertagging results, we now take the next step and evaluate the effect on end-to-end parsing. Apart from the issue of different error types having unpredictable effects, there are two other factors that make the isolated ubertagging results only an approximate indication of parsing performance. The first confounding factor is the statistical parsing disambiguation model. To show the effect of ubertagging in a realistic configuration, we only evaluate the first analysis that the parser returns. That means that when the unrestricted parser does not rank the gold analysis first, errors made by our model may not be visible, because we would never see the gold analysis in any case. On the other hand, it is possible to improve parser accuracy by pruning incorrect **lexitems** that were in a top ranked, non-gold analysis.

The second new factor that parser integration brings to the picture is the effect of resource limitations. For reasons of tractability, PET is run with per sentence time and memory limits. For treebank creation, these limits are quite high (up to four minutes), but for these experiments, we set the timeout to a more practical 60 seconds and the memory limit to 2048Mb. Without lexical pruning, this leads to approximately 3% of sentences not receiving an analysis. Since the main aim of ubertagging is to in-

Tag Type	ρ	F_1 Lexitem	F_1 Bracket	Time
No Pruning		94.06	88.58	6.58
FULL	0.00001	95.62	89.84	3.99
FULL	0.0001	95.95	90.09	2.69
FULL	0.001	95.81	89.88	1.34
FULL	0.01	94.19	88.29	0.64
INFL	0.0001	96.10	90.37	3.45
INFL	0.001	96.14	90.33	1.78
INFL	0.01	95.07	89.27	0.84
INFL	0.02	94.32	88.49	0.64
LTYPE	0.0002	95.37	89.63	4.73
LTYPE	0.002	96.03	90.20	2.89
LTYPE	0.02	95.04	89.04	1.23
LTYPE	0.05	93.36	87.26	0.88

Table 4: **Lexitem** and bracket F_1 over WSJ_{20}, with average per sentence parsing time in seconds.

crease efficiency, we would expect to regain at least some of these unanalysed sentences, even when a **lexitem** needed for the gold analysis has been removed.

Table 4 shows the parsing results at the same threshold values used in Table 3. Accuracy is calculated in terms of F_1 both over **lexitems**, and PARSEVAL-style labelled brackets (Black et al., 1991), while efficiency is represented by average parsing time per sentence. We can see here that an ubertagging F_1 of below 98 (cf. Table 3) leads to a drop in parser accuracy, but that an ubertagging performance of between 98 and 99 can improve parser F_1 while also achieving speed increases up to 8-fold.

From the table we confirm that, contrary to earlier pipeline supertagging configurations, tags of a finer granularity than LTYPE can deliver better performance, both in terms of accuracy and efficiency. Again, comparing graphically in Figure 7 gives a clearer picture. Here we have graphed labelled bracket F_1 against parsing time for the full range of threshold values explored, with the unpruned parsing results indicated by a cross.

From this figure, we see that the INFL model, despite being marginally less accurate when measured in isolation, leads to slightly more accurate parse results than the FULL model at all levels of efficiency.

Looking at the same graph for different samples of the development set (not shown) shows some

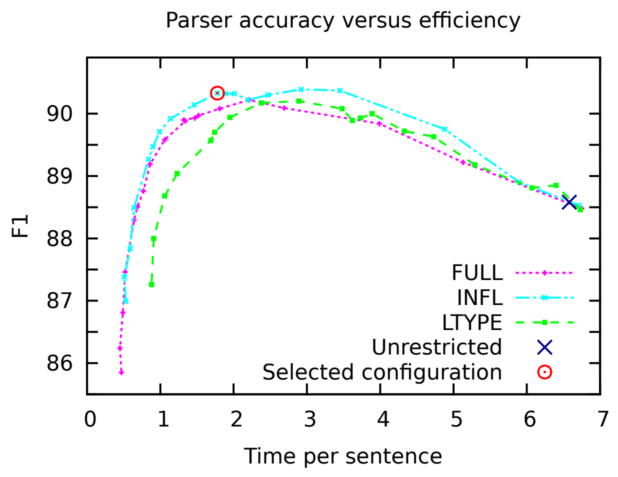


Figure 7: Labelled bracket F_1 versus parsing time per sentence over the development set, for each of the different ubertagging models. The cross indicates unpruned performance, while the circle pinpoints the configuration we chose for the final test runs.

variance in which threshold value gives the best F_1, but the relative differences and basic curve shape remains the same. From these different views, using the guideline of maximum efficiency without harming accuracy we selected our final configuration: the INFL model with a threshold value of 0.001 (marked with a circle in Figure 7). On the development set, this configuration leads to a 1.75 point improvement in F_1 in 27% of the parsing time.

8 Final Results

Table 5 shows the results obtained when parsing using the configuration selected on the development set, over our three test sets. The first, WSJ_{21} is from the same domain as the development set. Here we see that the effect over the WSJ_{21} set fairly closely mirrored that of the development set, with an F_1 increase of 1.81 in 29% of the parsing time.

The Wikipedia domain of our $WeScience_{13}$ test set, while very different to the newswire domain of the development set could still be considered in domain for the parsing and ubertagging models, since there is Wikipedia data in the training sets. With an average sentence length of 15.18 (compared to 18.86 in WSJ_{21}), the baseline parsing time is faster than for WSJ_{21}, and the speedup is not quite as large

	Baseline		Pruned	
Data Set	F_1	Time	F_1	Time
WSJ_{21}	88.12	6.06	89.93	1.77
$WeScience_{13}$	86.25	4.09	87.14	1.48
CatB	86.31	5.00	87.11	1.78

Table 5: Parsing accuracy in terms of labelled bracket F_1 and average time per sentence when parsing the test sets, without pruning, and then with lexical pruning using the INFL model with a threshold of 0.001.

but still welcome, at 36% of the baseline time. The increase is accuracy is likewise smaller (due to less issues with resource exhaustion in the baseline), but as our primary goal is to not *harm* accuracy, the results are pleasing.

The CatB test set is the standard out-of-domain test for the parser, and is also out of domain for the ubertagging model. The average sentence length is not much below that of WSJ_{21}, at 18.61, but the baseline parsing speed is still noticeably faster, which appears to be a reflection of greater structural ambiguity in the newswire text. We still achieve a reduction in parsing time to 35% of the baseline, again with a small improvement in accuracy.

The across-the-board performance improvement on all our test sets suggests that, while tuning the pruning threshold could help, it is a robust parameter that can provide good performance across a variety of domains. This means that we finally have a robust supertagging setup for use with the ERG that doesn't require heuristic shortcuts and can be reliably applied in general parsing.

9 Conclusions and Outlook

In this work we have demonstrated a lexical disambiguation process dubbed ubertagging that can assign fine-grained supertags over an ambiguous token lattice, a setup previously ignored for English. It is the first completely integrated supertagging setup for use with the English Resource Grammar, which avoids the previously necessary heuristics for dealing with ambiguous tokenisation, and can be robustly configured for improved performance without loss of accuracy. Indeed, by learning a joint segmentation and supertagging model, we have been able to achieve usefully high tagging accuracies for very fine-grained tags, which leads to potential parser speedups of between 4 and 8 fold.

Analysis of the tagging errors still being made have suggested some possibly avoidable inconsistencies in the grammar and treebank, which have been fed back to the developers, hopefully leading to even better results in the future.

In future work, we will investigate more advanced smoothing methods to try and boost the ubertagging accuracy. We also intend to more fully explore the domain adaptation potentials of the lexical model that have been seen in other parsing setups (see Rimell and Clark (2008) for example), as well as examine the limits on the effects of more training data. Finally, we would like to explore just how much the statistic properties of our data dictate the success of the model by looking at related problems like morphological analysis of unsegmented languages such as Japanese.

Acknowledgements

I am grateful to my colleagues from the Oslo Language Technology Group and the DELPH-IN consortium for many discussions on the issues involved in this work, and particularly to Stephan Oepen who inspired the initial lattice tagging idea. Thanks also to three anonymous reviewers for their very constructive feedback which improved the final version. Large-scale experimentation and engineering is made possible though access to the TITAN high-performance computing facilities at the University of Oslo, and I am grateful to the Scientific Computating staff at UiO, as well as to the Norwegian Metacenter for Computational Science and the Norwegian tax payer.

References

Srinivas Bangalore and Aravind K. Joshi. 1999. Supertagging: an approach to almost parsing. *Computational Linguistics*, 25(2):237 – 265.

Srinavas Bangalore and Aravind Joshi, editors. 2010. *Supertagging: Using Complex Lexical Descriptions in Natural Language Processing*. The MIT Press, Cambridge, US.

Ezra Black, Steve Abney, Dan Flickinger, Claudia Gdaniec, Ralph Grishman, Phil Harrison, Don Hindle, Robert Ingria, Fred Jelinek, Judith Klavans, Mark Liberman, Mitch Marcus, S. Roukos, Beatrice 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*, page 306 – 311, Pacific Grove, USA.

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

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

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

Stephen Clark and James R. Curran. 2007. Formalism-independent parser evaluation with CCG and DepBank. In *Proceedings of the 45th Meeting of the Association for Computational Linguistics*, page 248 – 255, Prague, Czech Republic.

Rebecca Dridan and Stephan Oepen. 2012. Tokenization. Returning to a long solved problem. A survey, contrastive experiment, recommendations, and toolkit. In *Proceedings of the 50th Meeting of the Association for Computational Linguistics*, page 378 – 382, Jeju, Republic of Korea, July.

Rebecca Dridan, Valia Kordoni, and Jeremy Nicholson. 2008. Enhancing performance of lexicalised grammars. page 613 – 621.

Rebecca Dridan. 2009. *Using lexical statistics to improve HPSG parsing*. Ph.D. thesis, Department of Computational Linguistics, Saarland University.

Murhaf Fares, Stephan Oepen, and Yi Zhang. 2013. Machine learning for high-quality tokenization. Replicating variable tokenization schemes. In *Computational Linguistics and Intelligent Text Processing*, page 231 – 244. Springer.

Murhaf Fares. 2013. ERG tokenization and lexical categorization: a sequence labeling approach. Master's thesis, Department of Informatics, University of Oslo.

Dan Flickinger, Yi Zhang, and Valia Kordoni. 2012. DeepBank. A dynamically annotated treebank of the Wall Street Journal. In *Proceedings of the 11th International Workshop on Treebanks and Linguistic Theories*, page 85 – 96, Lisbon, Portugal. Edições Colibri.

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

Taku Kudo, Kaoru Yamamoto, and Yuji Matsumoto. 2004. Applying conditional random fields to japanese morphological analysis. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, page 230 – 237.

Jonathan K. Kummerfeld, Jessika Roesner, Tim Dawborn, James Haggerty, James R. Curran, and Stephen Clark. 2010. Faster parsing by supertagger adaptation. In *Proceedings of the 48th Meeting of the Association for Computational Linguistics*, page 345 – 355, Uppsala, Sweden.

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

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

Kevin P. Murphy. 2002. Hidden semi-Markov models (HSMMs).

Stephan Oepen and John Carroll. 2000. Ambiguity packing in constraint-based parsing. Practical results. In *Proceedings of the 1st Meeting of the North American Chapter of the Association for Computational Linguistics*, page 162 – 169, Seattle, WA, USA.

Stephan Oepen, Daniel Flickinger, Kristina Toutanova, and Christopher D. Manning. 2004. LinGO Redwoods. A rich and dynamic treebank for HPSG. *Research on Language and Computation*, 2(4):575 – 596.

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

Laura Rimell and Stephen Clark. 2008. Adapting a lexicalized-grammar parser to contrasting domains. page 475 – 484.

Kristina Toutanova and Colin Cherry. 2009. A global model for joint lemmatization and part-of-speech prediction. In *Proceedings of the 47th Meeting of the Association for Computational Linguistics*, page 486 – 494, Singapore.

Gisle Ytrestøl. 2012. *Transition-based Parsing for Large-scale Head-Driven Phrase Structure Grammars*. Ph.D. thesis, Department of Informatics, University of Oslo.

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

Yue Zhang and Stephen Clark. 2010. A fast decoder for joint word segmentation and POS-tagging using a single discriminative model. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, page 843 – 852, Cambridge, MA, USA.

Yi Zhang, Stephan Oepen, and John Carroll. 2007. Efficiency in unification-based n-best parsing. In *Proceedings of the 10th International Conference on Parsing Technologies*, page 48 – 59, Prague, Czech Republic, July.