Why Supertagging Is Hard

François Toussenel
Lattice – UMR 8094
University Paris 7
francois.toussenel@linguist.jussieu.fr

Abstract

Tree adjoining grammar parsers can use a supertagger as a preprocessor to help disambiguate the category[1] of words and thus speed up the parsing phase dramatically. However, since the errors in supertagging propagate to this phase, it is vital to keep the error rate of the supertagger phase reasonably low. With very large tagsets coming from extracted grammars, this error rate can be of almost 20%, using standard Hidden Markov Model techniques. To combat this problem, we can trade a higher precision for increased ambiguity in the supertagger output. I propose a new approach to introduce ambiguity in the supertags, looking for a suitable trade-off. The method is based on a representation of the supertags as a feature structure and consists in grouping the values, or a subset of the values, of certain features, generally those hardest to predict.

1 Introduction

This paper deals with supertagging[2] as a preprocessing step before full parsing.

A TAG parser has too many elementary trees to choose from if they are not at least partially disambiguated beforehand (Joshi and Bangalore, August 1994): the combinatorics at the parsing level are huge. As suggested in Srinivas Bangalore's Ph.D. thesis (Bangalore, 1997), supertagging may be used to reduce the high number of trees associated with each word. But to tag and parse real-world text, we need a sufficiently sized grammar. One convenient way to constitute a large TAG is to extract it from a hand-corrected treebank. Naturally, the resulting tagset for supertagging is also large. The problem thus becomes the fact that when the tagset is very large (e.g. about 5,000 different trees), the precision of the supertagger output is so low (about 80%) that the parser fails on most sentences.

The supertagger we use is based on a Hidden Markov Model (HMM) tagger trained on a grammar extracted (Chen, 2001) from the Wall Street Journal part of the Penn Treebank (Marcus et al., 1993) and the parser is the one described in (Nasr et al., 2002).

2 Supertagging and Very Large Tagsets

If HMM part of speech tagging has been proven quite successful, supertagging is more problematic for two main reasons.

- (A) The large number of categories which characterizes supertagging entails statistical problems, but for the result to be useful in helping parse real-world texts, a medium-sized or small grammar (with e.g. 300 or 400 different elementary trees) seems insufficient.
- (B) The non-local nature of the information included in the supertag clashes with the local vision of the HMM tagger (e.g. a three-word window). Indeed, supertags locally represent dependencies not represented in parts of speech. For instance, the supertag assigned to the verb *brought* in *I brought their children my son's old bicycle* will include a slot for each of the two complements, the second of which (*my son's old bicycle*) is beyond the three-word window in this sentence.

With a tagset of about 5,000 trees, HMM tagging techniques suffer from severe training data sparseness. Statistical problems arise that are little or not encountered in a regular part of speech tagging context. Indeed, various types of events are never seen in the training corpus. The simplest type is the supertag itself. Some supertags are new in the test corpus. Obviously, standard techniques cannot guess them.

[1] Specifically, a rich description of the syntactic properties of words.

[2] Supertagging consists in assigning an elementary tree (of a TAG) to each word of a sentence.

TAG+7: Seventh International Workshop on Tree Adjoining Grammar and Related Formalisms.
May 20-22, 2004, Vancouver, BC, CA.
Pages 96-103.

A more frequent type of unseen events is the association of known words with known supertags that did not occur together in the training corpus. About 5% of the word-supertag pairs are new, these pairs being involved in about a quarter of the errors[3]. John Chen (Chen, 2001) has addressed this problem and has designed tree families to automatically extend the grammar. In (Hockenmaier and Steedman, 2002), a Combinatory Categorial Grammar is extracted from the Penn Treebank and the authors have found a 26% reduction of "unseen pairings of seen words and seen categories" (from 3% to 2.2%) thanks to a reduction of the category inventory, and a 50% reduction (from 3% to 1.5%) when combining the reduced category inventory with a more elaborate treatment of unknown words (using the part of speech instead of a single token for unknown words).

Other existing solutions include reranking (Chen et al., 2002) and class tagging (Chen et al., 1999) (Chen, 2001), but either they are applied to smaller grammars (between 300 and 500 different trees) or they face problems similar to ours.

The reranking technique notably is not bound to a limited context and is thus complementary with an n-gram tagger.

3 Ambiguous Supertags

Failing to find the correct supertag often enough for the parse to succeed, we resort to allowing some ambiguity in the supertagger output. The main idea is to relieve the supertagger from a part of its disambiguating duty, deferring it to the parser which will make the final decisions (given that it has information about the whole sentence). The key point is finding a good trade-off between precision rate (for successful parses) and ambiguity (to keep the parsing phase tractable).

With the n-best tagging technique (Bangalore and Joshi, 1999), the supertagger outputs several trees (the most probable n supertags) and the parser chooses among them. One drawback is that the output consists in the same number of supertags for each word, regardless of its type (e.g. verb or adjective), whereas it seems attractive to keep more possible supertags for a verb than for less ambiguous words, for instance.

Previously we tested a kind of n-best supertagging on our grammar, but failed to achieve an error rate below 9.5%, which was unsatisfactory and led us to imagine harder ways to produce an ambiguous output.

[3]The results presented here have been computed from a supertagged portion of the Penn Treebank consisting of 1,939 sentences (about 50K words), the training corpus consisting of 37,858 sentences (about 980K words).

3.1 Underspecification Using a Feature Structure

The solution I propose introduces underspecification at the supertag level. In other words, the supertag conveys less information, but still more than in mere parts of speech. To do this I represent the trees as a feature structure in which the salient characteristics of a supertag are encoded, as was initially suggested in John Chen's Ph.D. (for another purpose) (Chen, 2001)[4].

The results presented here are from experiments using a structure of 18 features, among which are:

- the part of speech of the root node (26 possible values),
- the subcategorization (more than one hundred possible values),
- several transformational features,
- the two ordered lists of the nodes on the left and right frontiers,
- the list of internal nodes (neither the root nor the nodes on the frontier),
- the list of co-anchors (more than one hundred possible values),
- the part of speech of the modified word if this is a modifier,
- the direction of the modification if this is a modifier.

3.2 More on Two Features

Two features are of particular interest (both are pertinent only for modifier trees): one specifies the part of speech of the modified word and the other specifies the direction of the modification. It must be noted that both these features have an extra value (*NIL*) which means non-pertinent, for the case of a non-modifier word: thus predicting this feature involves predicting whether the word is a modifier. These are the most difficult features to predict (error rates of about 12.6% for the first with 38 possible values and almost 9% for the second with only 3 possible values). Moreover, predicting them makes the supertagging process much longer. However, as is shown below, knowing their values for a given supertag helps predict other features, including the part of speech.

3.3 Neutralizing Features

By neutralizing certain features describing the trees (i.e. not specifying the value for those features), we obtain an underspecified supertag (the tagset is therefore reduced), which is thus ambiguous but easier to predict.

[4]For my experiments I used John Chen's feature structures but my plans for future work involve the use of others.

This approach allows us to control the amount of information we are able and willing to supply the parser with[5]. This is particularly interesting since the error comes from a relatively small number of features each time (but the features which are incorrectly predicted are not always the same). Table 1 shows that 42% of the errors on trees[6] involve only up to two features.

Table 1: Cumulated proportion or errors due to n features incorrectly predicted (the remaining 6.3% is due to co-anchors).

# of features	% of errors (cumulated)
1	19.972
2	42.038
3	54.429
4	62.934
5	74.584
6	82.122
7	85.440
8	89.660
9	91.485
10	92.290
11	93.386
12	93.611
13 to 18	93.697

It is important to state that the feature neutralization must take place only after training and supertagging. Indeed, if the supertagger is trained on an "underspecified" annotated corpus, it gives worse results than if it is trained on a corpus annotated with regular supertags, its output then being modified to change the regular supertags into their underspecified versions. For instance, there is a 15% relative reduction of the error rate for the part of speech feature when we tag the whole supertag. This is due to the dependencies between the features: learning on more features helps predict one particular feature. Of course, if it is just to tag with part of speech, the whole process takes much more time than regular part of speech tagging. On the other hand, the precision is higher (the two features mentioned above are in a large part responsible for this).

3.4 Experiments on (Almost) All the Combinations

As a first trial in this direction I conducted experiments consisting in neutralizing series of sets of features to study the coordinated behavior of both the error rate and ambiguity according to the features neutralized. The combinatorics are rather large[7], but evaluating the error rate and ambiguity of the supertagged output with a given set of neutralized features takes very little time (about one or two seconds on a personal computer). Indeed, since the input text is tagged with the full supertag, there is no need to supertag the text for each set of neutralized features. We only have to extract the reduced information from the supertags in both the hypothesis and the reference and run the evaluation on it.

I decided never to neutralize the part of speech feature (numbered 0); I thus gathered the resulting 131,072 error rate/ambiguity pairs. To find a good trade-off between error rate and ambiguity, one needs to consider some candidate sets of neutralized features; a simple method to preselect some candidates is to search for the set of neutralized features yielding the lowest ambiguity for a (small) number of given maximum error rates. Figure 1 and Table 2 show the lowest ambiguity for each given maximum entire error rate, from 19% to 4%. These boundaries come from the error rate associated with no neutralization at all (18.64%) and the one associated with all the features neutralized except part of speech (3.67%).

The ambiguity figures are the average number of supertags (from the original tagset) represented by the underspecified tag for each word in the test corpus. Thus it depends on the tags chosen for each word, it is not just the ambiguity of the simplified grammar with regard to the original grammar.

The lowest error rate is a bit under 4%, associated with an average ambiguity above 450, and corresponds to a tag representing (a little more than) part of speech. We hope to find at least one set of neutralized features allowing for acceptable error rate and ambiguity. Error rates of 6% or 5% would seem suitable, but they are associated with an ambiguity of about 212 and 306 respectively. It is not sure that such high ambiguity can be handled by a processor with such input; in the case of a statistical parser, the resulting combinatorics would make it necessary to use an appropriate beam search. However, the accuracy of the parser is not guaranteed to be preserved with such a beam search.

3.5 The Incremental Method

Exploring the whole set of possible combinations is affordable when each test mainly involves translating tags. However, the performance of the supertagger in itself is not the only relevant measure when it comes to use its output as an input for a parser. To find the best trade-off between error rate and ambiguity, the most natural test is the performance of the parser. We would need both the accuracy of its output and the average time it takes to parse a sentence. These experiments are yet to be done,

[5] To do this we can neutralize certain features altogether or tag with a set of values for certain features instead of only one value for those features.

[6] not including errors on a co-anchor.

[7] A structure of 18 features entails $2^{18} = 262144$ possible sets of neutralizations.

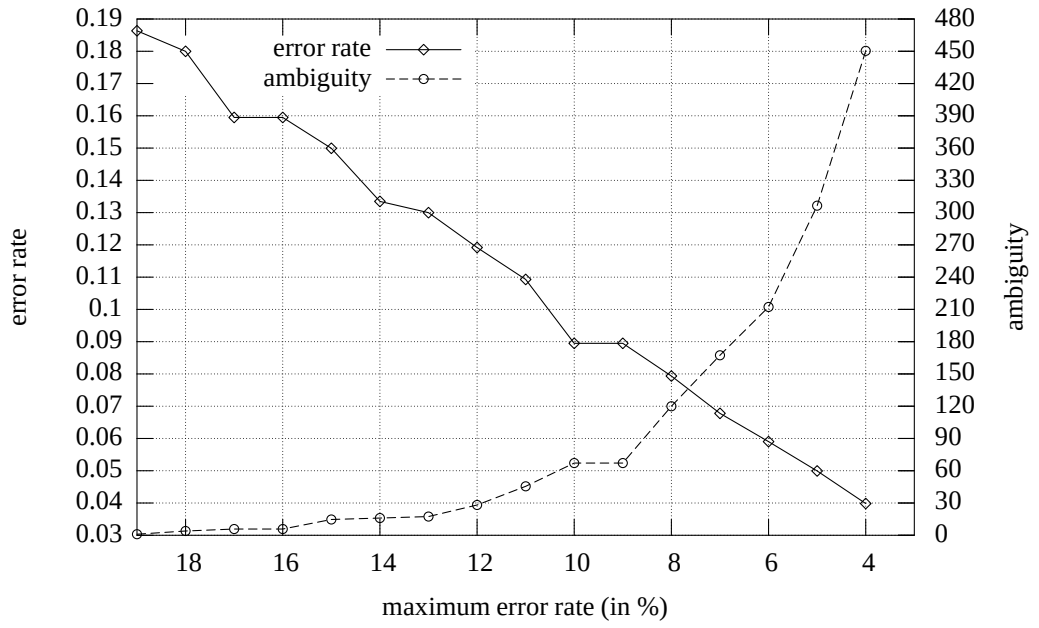

Figure 1: Lowest ambiguity per WER (from full set of combinations).

Table 2: Lowest ambiguity per WER (from full set of combinations). The third column (#) represents the number of neutralized features for the given set (detailed in the fourth column). The word error rate is given in %.

WER	Ambig.	#	Neutralized features
18.635	1.014	1	9
17.998	3.944	5	1 2 6 11 15
15.948	5.810	1	16
15.948	5.810	1	16
14.993	14.699	2	11 16
13.347	16.012	2	16 17
12.999	17.394	10	3 4 6 7 8 12 13 15 16 17
11.917	28.162	10	2 3 5 7 8 12 14 15 16 17
10.928	45.532	3	11 16 17
8.950	67.129	6	1 2 11 13 16 17
8.950	67.129	6	1 2 11 13 16 17
7.935	119.928	8	1 2 11 13 14 15 16 17
6.778	167.370	9	1 2 3 11 12 13 14 16 17
5.903	212.301	10	1 2 3 11 12 13 14 15 16 17
4.994	306.446	13	1 2 3 4 7 9 11 12 13 14 15 16 17
3.984	450.421	15	1 2 3 4 5 6 7 9 11 12 13 14 15 16 17

but one can already guess that the same thorough series of tests will probably not be tractable when testing the parser every time. Not only the tests will take the time of parsing, but this time will increase exponentially with the ambiguity. As a matter of fact, achieving a test in a limited time is a result in itself: it means the parser can handle the ambiguity.

Consequently, the tests must be run in the order of fastest to slowest. Also, we will not run all the tests but only those that have the best chances to reveal interesting results. That is, we need a method to select the combinations.

With this objective in mind, I designed an incremental method to choose which features are to be neutralized in order to minimize the error rate. I applied this method to the supertagger output, which gives a preview of its usefulness (I hope it will be as useful with the parser evaluation). There again, the result is a graduated trade-off between precision and ambiguity. We will compare it to our previous combinations.

I now describe the incremental method as I applied it on the supertagger output. The goal is to construct a number of sets of neutralized features, from a set of one feature to a set of 17 features (for a structure of 18 features). The main idea is follow the optimum "path" by selecting the most interesting feature to neutralize at each step, adding it to the previous set. Let S be the current set of neutralized features. I first decided to always keep the feature representing the part of speech of the anchor of the tree. So the second step was to add one of the 17

remaining features to the (yet empty) set S. To choose this feature, each of the candidate features is temporarily added to S and the corresponding error rate and ambiguity are computed. The feature leading to the best result is then selected and permanently added to S. The process is repeated with the remaining features until there are no more features to neutralize and only the part of speech (which is our baseline) remains.

The number of tests required by this method is only $\sum_{i=1}^{17} i = 154$ instead of $2^{17} = 131,072$ for the full set of combinations.

The search for the next feature to neutralize can be driven by three types of criteria: the error rate, the ambiguity, or a combination of the two. I tried the first two criteria, which selected different features but yielded similar trade-offs.

3.6 Experiments on the Incremental Method Applied to the Supertagger

Figure 3 shows the linked progression of the error rate and ambiguity, using the error rate criterium to select each feature to neutralize. Here the relevant curves are those marked as *feature*. We will see *values* below.

It is interesting to see how the incremental method behaves compared with the full set of combinations. Figure 2 compares the two corresponding curves (features being either fully neutralized or not at all). Let us first note that the incremental method's curve is very similar to the curve obtained by selecting the lowest error rate per number of neutralized features, as opposed to the average error rates per number of neutralized features. Indeed, only 4 out of 17 sets of neutralized features are different in the two curves.

Let's take a closer look at those four couples of sets in Table 3. What happened is that for the lowest error rate from the full set of combinations, the set of 6 neutralized features (1 2 11 13 16 17) has only 3 features (11 16 17) in common with the set of 5 neutralized features (11 12 14 16 17). Of course, the incremental method cannot compete with this performance because it keeps the whole previous set by design.

What's more, both the error rate and the ambiguity are lower for the set of 6 features than for the set of 5 features. For the other sets, the balance between error rate and ambiguity is regular again, and new features are just added to the previous set, just like in the incremental method.

The combinations of error rate and ambiguity drop from 18.64%/1 with no neutralization at all to 3.67%/509 for just part of speech. The point of the method is to choose an intermediate value (the best trade-off). For example, with 11 neutralized features, we have 5.17%/284, and for 10 neutralized features, 5.9%/212.

3.7 Refinement

A slight improvement of this method can be achieved by neutralizing only part of the features as opposed to the features altogether. In other words, instead of grouping all the values for a given feature, we can group some values. For instance, consider the three-value feature *direction of modification*. The three possible values are *left*, *right* or *NIL* (in case this is not a modifier). We could group the first two values, which would result in a binary feature simply indicating whether this is a modifier. The selection of values to group can be done according to error analysis. We group the values which the supertagger most often confuses.

To evaluate the power of this improved method, I used the same test corpus for both the error analysis and the new evaluation with grouped values, which one cannot do when (super)tagging new text but this shows the maximum gain we can get thanks to this refined method.

On Figure 3, the relevant curves are those marked as *values*. The error rate curve is the same as the old one, since all values which were confused by the supertagger on this test corpus, and only these values, were grouped. Only ambiguity is different, and naturally always lower or equal, since the supertags represented by the underspecified tags have all their features present, only with ambiguous values for some of them.

To compare with the previous results, with 11 neutralized features, ambiguity drops from 284 down to 248, and for 10 features, from 212 down to 185.

As we can see, the ambiguity associated with acceptable error rates is still quite large, even with the refined method. This seems to indicate that this kind of approach is not sufficient. Replacing the error rate criterium with the parser's accuracy, as was explained above, will probably highlight better trade-offs, but it seems likely that the improvement will be limited.

3.8 Feature Structures

All my experiments were based on John Chen's various feature structures. I believe the choice of a feature structure must have a noticeable (but somewhat limited) influence on the results one can get from playing with ambiguity in the way described in this paper. But there is more to it: the extracted grammar itself is determined by the feature structure. The grammar I used is very close to what was seen in the training corpus. A good deal of generalization can be done, though, and this would probably entail lower error rates for the same amount of ambiguity. Metagrammars (Candito, 1999) and their associated feature structures are designed in this spirit. First, features are drawn, then a generalization phase takes place, and finally the grammar is extracted. Thus unseen supertags are less likely to appear in a test corpus.

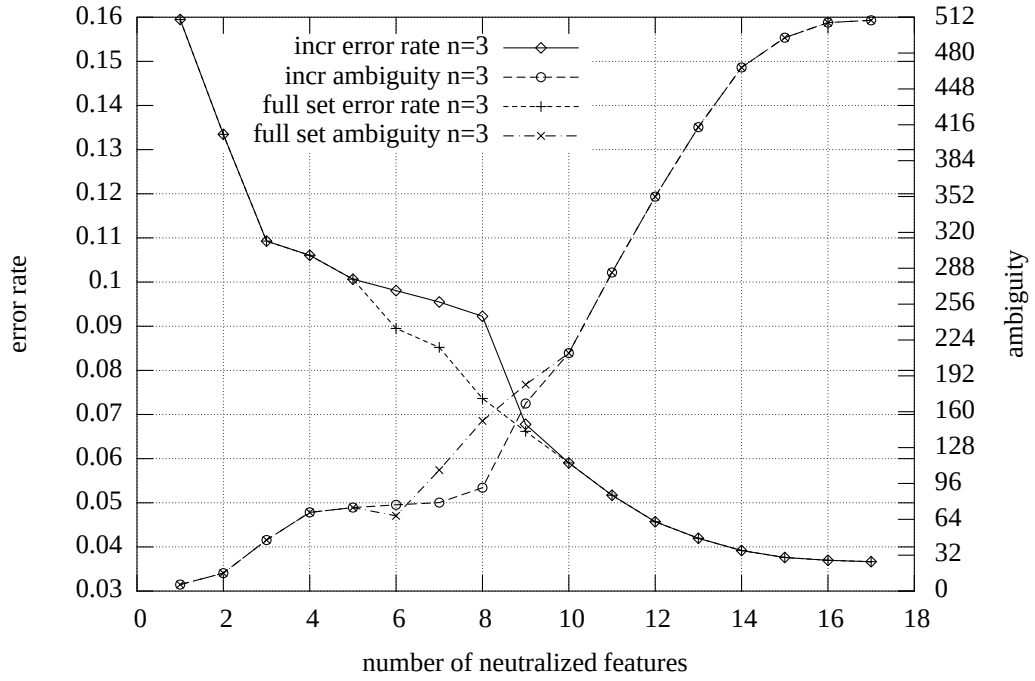

Figure 2: Linked progression of error rate and ambiguity, for whole *feature* neutralization, to compare the incremental method with the full set.

Table 3: Different sets between incremental method and whole combination (lowest error rate).

	Whole combination			Incremental		
#	WER	Ambig.	Neutralized features	WER	Ambig.	Neutralized features
6	8.95	67.129	1 2 11 13 16 17	9.81	76.959	16 17 11 14 12 3
7	8.52	107.972	1 2 11 13 14 16 17	9.55	78.925	16 17 11 14 12 3 2
8	7.36	151.844	1 2 11 12 13 14 16 17	9.22	92.238	16 17 11 14 12 3 2 1
9	6.62	184.199	1 2 11 12 13 14 15 16 17	6.78	167.370	16 17 11 14 12 3 2 1 13

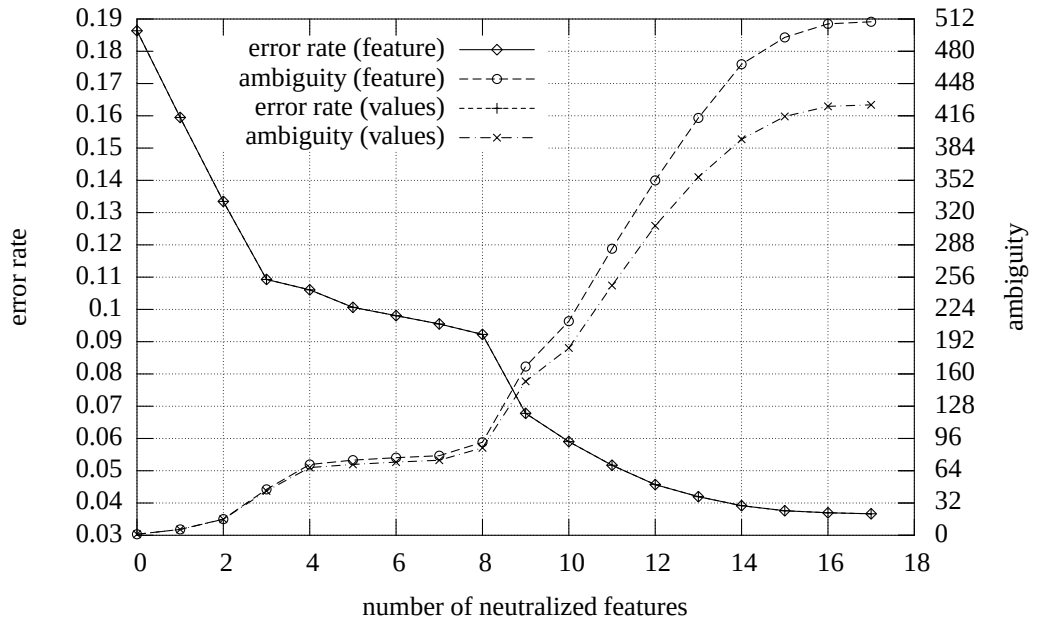

Figure 3: Linked progression of error rate and ambiguity, using only the incremental method, to compare whole *feature* neutralization and selected *values* neutralization. The error rate is the same in both cases. Here the selection of values is driven by error analysis on the same test corpus (to show the theoretical maximum gain we could get with this precise method).

4 Conclusion

The incremental method, while not being perfect, can offer a good approximation at a low cost.

Having applied various Hidden Markov-derived models on supertagging with large extracted grammars, I believe that with such a large tagset it is impossible to achieve a precision rate acceptable for parsing in a single process. Consequently, underspecification imposes itself as one of the most promising directions in this respect. Hopes for future work on this subject mainly lie in a grammar less dependent on the treebank from which it is extracted, in a feature structure better structured (using Metarules (Xia, 2001) or inspired by (Kinyon, 2000) which rely on a Metagrammar (Candito, 1999)), and more importantly in a shallow parsing phase eliminating supertags which would not fit in, thanks to a global consideration of the sentence.

In this last respect, it must be noted that many supertagged sequences are inconsistent: I have observed that a third of them contained at least a supertag which required a certain category before or after it that was not in the relevant part (either to the left or to the right) of the sequence. It is clear that a global vision of the sentence can help reduce the ambiguity of the supertags. The difficulty is to keep the computation simple and fast enough to be used efficiently before full parsing.

Acknowledgments

I wish to particularly thank Owen Rambow, Alexandra Kinyon and Laura Kallmeyer for their precious help with the abstract and the final version of this paper.

References

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

Srinivas Bangalore. 1997. *Complexity of lexical descriptions and its relevance for partial parsing*. Ph.D. thesis, University of Pennsylvania, Philadelphia.

Marie-Hélène Candito. 1999. *Représentation modulaire et paramétrable de grammaires électroniques lexicalisées*. Ph.D. thesis, University Paris 7.

John Chen, Srinivas Bangalore, and K. Vijay-Shanker. 1999. New models for improving supertag disambiguation. In *Proceedings of the Ninth Conference of the European Chapter of the Assocation for Computational Linguistics*, Bergen, Norway.

John Chen, Srinivas Bangalore, Michael Collins, and Owen Rambow. 2002. Reranking an n-gram supertagger. In *Proceedings of the Sixth International Workshop on Tree Adjoining Grammars and Related Frameworks*, Venice, Italy.

John Chen. 2001. *Towards Efficient Statistical Parsing using Lexicalized Grammatical Information*. Ph.D. thesis, University of Delaware.

Julia Hockenmaier and Mark Steedman. 2002. Acquiring Compact Lexicalized Grammars from a Cleaner Treebank. *Proceedings of Third International Conference on Language Resources and Evaluation (LREC)*.

Aravind K. Joshi and Srinivas Bangalore. August 1994. Disambiguation of super parts of speech (or supertags): Almost parsing. In *International Conference on Computational Linguistics (COLING 94)*, Kyoto University, Japan.

Alexandra Kinyon. 2000. Hypertags. In *Proceedings of COLING-00*, Saarbrücken, Germany.

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

Alexis Nasr, Owen Rambow, John Chen, and Srinivas Bangalore. 2002. Context-Free Parsing of a Tree Adjoining Grammar Using Finite State Machines. In *Sixth International Workshop on Tree Adjoining Grammars and Related Frameworks*, Venice, Italy.

Fei Xia. 2001. *Automatic grammar generation from two different perspectives*. Ph.D. thesis, Department of Computer and Information Science, University of Pennsylvania.