SUPERTAGGING: A NON-STATISTICAL PARSING-BASED APPROACH

Pierre Boullier
INRIA-Rocquencourt
Domaine de Voluceau
B.P. 105
78153 Le Chesnay Cedex, France
Pierre.Boullier@inria.fr

Abstract

We present a novel approach to supertagging w.r.t. some lexicalized grammar G. It differs from previous approaches in several ways:

- These supertaggers rely only on structural information: they do not need any training phase;
- These supertaggers do not compute the "best" supertag for each word, but rather a set of supertags. These sets of supertags do not exclude any supertag that will eventually be used in a valid complete derivation (i.e., we have a recall score of 100%);
- These supertaggers are in fact true parsers which accept supersets of $\mathcal{L}(G)$ that can be more efficiently parsed than the sentences of $\mathcal{L}(G)$.

1 Introduction

In [7], Joshi and Srinivas introduced the idea of supertagging in order to improve the efficiency of parsers for lexicalized formalisms by selecting, for each word, its appropriate descriptions given its context in a sentence. In NLP, numerous grammatical formalisms are lexicalized. In this paper, we will concentrate on supertagging of lexicalized tree-adjoining grammars (LTAGs) (See [8]). In LTAGs, each elementary tree contains at least one lexical item called an *anchor*. All the arguments of this anchor are instantiated at places (nodes) through either a substitution or an adjunction operation. Thus an elementary tree may be seen as a description of the context, the domain of locality, of its anchor, including long distance dependencies. Since an elementary tree (either initial or auxiliary) defines its anchor precisely, it is called a *supertag*, because it conveys much more information than the standard part-of-speech tag. As a consequence, in the LTAG context, the lexical ambiguity of a word (i.e., the number of supertags associated with it) is generally much greater than its number of standard part-of-speech tags.

Many LTAG parsers work in two phases: for each word in a sentence, the first phase selects the appropriate supertags, while the second phase combines the selected supertags through substitutions and adjunctions. The first phase is called *supertagging*. *Supertag disambiguation* is the process by which local lexical ambiguity can be reduced or eventually even resolved, resulting in a single supertag per word. This selection of the most probable supertag is usually performed using statistical distributions of supertag co-occurrences extracted from (large) annotated corpora of parses. In this context, the result of a supertagger is almost a parse in the sense that a parser only needs to link together its supertags into a single structure. If such a single structure cannot be built, we have a partial parser (See, for example, [9]). However, this

approach which assigns a single supertag per word, even if it is extended to produce the n-best supertags for each word, may well eliminate trees which are parts of valid parses (a detailed discussion on supertagging can be found in [10]).

In this paper we will depart from the supertagging community on several points:

1. non-statistical;

2. strictness;

3. parsing-based.

The first point means that, in our approach, supertag disambiguation will not be done in a statistical setting, thus avoiding the need for training on annotated corpora, which are very costly to develop.

The second point means that we adopt a *strict* supertag disambiguation approach. A strict supertagger cannot eliminate supertags that could be parts of some complete parse.

In contrast to Srinivas and Joshi who estimate in [10] that only "*local techniques can be used to disambiguate supertags" and that "full parsing is contrary to the spirit of supertagging*", in point three, we advocate that a supertagger can indeed be a parser. A potential advantage of this approach is that our supertaggers may use global information to take their decisions while only local (statistical) information (n-gram model) is used in traditional supertaggers.[1] However, this parser must be "sufficiently" efficient. This means, of course, that it cannot be based upon the original LTAG. We thus assume that a supertagger relies upon a grammar which is not the original LTAG. For evident reasons of reliability and development costs, we require that the grammars upon which our supertaggers are based must be automatically deduced from the original LTAG. In other words, our purpose is to reduce supertag ambiguity by using only structural information which can be automatically extracted from any given LTAG.

This paper is divided into three main parts. In Sections 2 and 3, we study how it is possible to define a context-free (CF) superset and a regular superset for any given tree-adjoining language (TAL). In Sections 4 and 5, we study how general recognizers for these supersets can be transformed into supertaggers for the original LTAG. Finally, in Section 7 we report on the experiments that have been performed with a wide-coverage English LTAG, on a large test set.

2 From TAG to CFG

The purpose of this section is to show how any TAG G that defines the TAL $\mathcal{L}(G)$ can be transformed into a CF grammar (CFG) G_{cf} whose language $\mathcal{L}(G_{\text{cf}})$ is a superset of $\mathcal{L}(G)$. The process shown here is a direct transformation from TAG to CFG and is reminiscent of Boullier's work which proposes such a transformation, using the intermediate formalism of range concatenation grammars (RCG).[2]

[1] The *dependency model* of Srinivas and Joshi in [10] does not put any limitation on the size of the window. However, they stopped their experiments because "*[they] are restrained by the lack of a large corpus of LTAG parsed derivation structures that is needed to reliably estimate the various parameters of this model*".

[2] Following [3] G, is first transformed into an equivalent simple positive RCG (PRCG). Then, following [1], this simple PRCG is transformed into a simple 1-PRCG which defines a CF superset. This simple 1-PRCG is in turn transformed into an equivalent CFG (See [2]).

We assume here that the reader is familiar with the TAG formalism (See [6] for an introduction). Let $G = (V_N, V_T, \mathcal{I}, \mathcal{A}, S)$ be a TAG,[3] we will see how each initial tree of $\mathcal{I}$ and each auxiliary tree of $\mathcal{A}$ can be transformed into CF productions to form a CFG $G_{\rm cf} = (N, T, P, S)$ such that $\mathcal{L}(G) \subset \mathcal{L}(G_{\rm cf})$.

G and $G_{\rm cf}$ have the same set of terminal symbols (i.e., $V_T = T$) and the same start symbol S. The set N of nonterminal symbols of $G_{\rm cf}$ is defined by $N = V_N^{\mathcal{I}} \cup \{A_i \mid i \in \{L, R\} \text{ and } A \in V_N^{\mathcal{A}}\}$. That is, nonterminal symbols which label substitution nodes and the roots of initial trees, are kept unchanged, while the nonterminal symbols, say A, which label adjunction nodes, roots and feet of auxiliary trees, produce two CF nonterminal symbols A_L and A_R. The intended meaning of A_L is to capture the set of strings generated by the parts of the auxiliary A-trees which lie to the left of their spines, while A_R will capture the set of strings generated by the parts of the auxiliary A-trees which lie to the right of their spines.

As in [3], for each elementary tree τ, its *decoration string* σ_τ is an element of $(N \cup T)^*$ defined as follows. We perform on τ a top-down left-to-right traversal that collects the labels of the traversed nodes in σ_τ. The traversals (both top-down and bottom-up) of root nodes in initial trees leave no trace in σ_τ. For an auxiliary A-tree τ, the top-down traversal of its root node initializes σ_τ with A_L. Upon completion of this traversal, the bottom-up visit of the root node of τ will eventually complete σ_τ with A_R. Except for foot nodes, inside nodes and leaf nodes of initial and auxiliary trees are equally processed. The traversal of a leaf node labeled l results in appending l to σ_τ (if $l = \varepsilon$, σ_τ is left unchanged). The top-down traversal of an inside (adjunction) node labeled A results in appending A_L to σ_τ while its bottom-up traversal results in appending A_R to σ_τ. The (left-to-right) traversal of a foot node labeled A results in appending $A_L A_R$ to σ_τ. If τ is an auxiliary A-tree, let σ_τ^L be the prefix of σ_τ collected before the traversal of its foot node (including A_L) and let σ_τ^R be the suffix of σ_τ collected after the traversal of its foot node (starting with A_R).[4]

If τ is an initial A-tree and if $\sigma_\tau = X_1 \ldots X_p$ is its decoration string, we associate to τ the single CF production

$$A \rightarrow X_1 \ldots X_p$$

If τ is an auxiliary A-tree and if $\sigma_\tau = \sigma_\tau^L \sigma_\tau^R$ is its decoration string, we associate to τ the pair of CF productions

$$\begin{aligned} A_L &\rightarrow X_1 \ldots X_p \\ A_R &\rightarrow Y_1 \ldots Y_q \end{aligned}$$

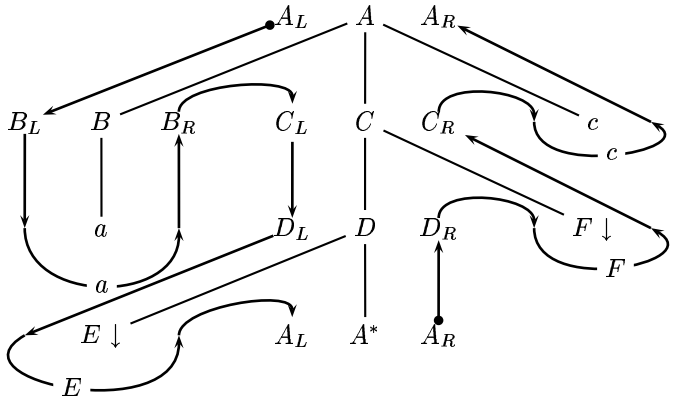
if we have $\sigma_\tau^L = X_1 \ldots X_p$ and $\sigma_\tau^R = Y_1 \ldots Y_q$. Moreover, by construction (and without any adjunction constraints), we have $p \geq 2$, $X_1 = X_p = A_L$ and $q \geq 2$, $Y_1 = Y_q = A_R$.[5]

Note that the same CF production can be produced by different auxiliary trees.

[3] An A-tree is an elementary tree whose root node is labeled by the nonterminal symbol $A \in V_N$. We assume that the nonterminal symbols $V_N^{\mathcal{I}}$ which label the roots of the initial trees are disjoint from the nonterminal symbols $V_N^{\mathcal{A}}$ which label the roots of the auxiliary trees, we have $V_N^{\mathcal{I}} \cap V_N^{\mathcal{A}} = \emptyset$ and $V_N^{\mathcal{I}} \cup V_N^{\mathcal{A}} = V_N$, that the start symbol S is an element of $V_N^{\mathcal{I}}$, that each internal node of $\mathcal{I} \cup \mathcal{A}$ is labeled by an element of $V_N^{\mathcal{A}}$, while each leaf of $\mathcal{I} \cup \mathcal{A}$ is labeled by an element of $V_N^{\mathcal{I}} \cup V_T \cup \{\varepsilon\}$, except that each foot node bears the same label as its root node.

[4] If a node has a null adjunction constraint, its top-down and bottom-up traversals (or left-to-right traversal for a foot node) leave the decoration string unchanged.

[5] These productions are both left and right recursive and thus, $G_{\rm cf}$ is ambiguous.



$$\sigma_\tau = \underbrace{A_L\, B_L\, a\, B_R\, C_L\, D_L\, E\, A_L}_{\sigma_\tau^L}\ \underbrace{A_R\, D_R\, F\, C_R\, c\, A_R}_{\sigma_\tau^R}$$

$$\begin{aligned} A_L &\rightarrow A_L\ B_L\ a\ B_R\ C_L\ D_L\ E\ A_L \\ A_R &\rightarrow A_R\ D_R\ F\ C_R\ c\ A_R \end{aligned}$$

Figure 1: Translation of an auxiliary A-tree τ into a pair of CF productions

It is not hard to see that, as such, the nonterminal symbols of the form A_L and A_R cannot generate any terminal string. This can be amended by adding to P, for each $A \in V_N^A$, the pair of ε-productions

$$\begin{aligned} A_L &\rightarrow \varepsilon \\ A_R &\rightarrow \varepsilon \end{aligned}$$

which will terminate any sequence of adjunction operations.

To illustrate this process, in Figure 1, we have the translation of an auxiliary A-tree τ into a pair of CF productions.

It is not difficult to see that if $w \in \mathcal{L}(G)$, we have $w \in \mathcal{L}(G_{\text{cf}})$. More precisely, if we consider the decoration string σ associated with any derived tree that can be composed, starting from an initial S-tree, this string can be derived from S w.r.t. G_{cf} (i.e., $S \underset{G_{\text{cf}}}{\overset{+}{\Rightarrow}} \sigma$).[6] Moreover, if this derived tree is complete (no more substitution can take place) and if its terminal yield is w (that is $w \in \mathcal{L}(G)$), we have $\sigma \underset{G_{\text{cf}}}{\overset{*}{\Rightarrow}} w$.

If G is an LTAG, it does not mean that its associated CFG G_{cf} is also lexicalized: the productions generated by initial trees are lexicalized, at least one production in the pair of productions generated by auxiliary trees is lexicalized, while the ε-productions are surely not lexicalized.

[6] After substitution at some leaf node, the root node of an initial tree becomes an inside node of the derived tree, we assume that the traversal of such an inside node leaves no trace in σ.

3 From TAG to FA

In this section, we examine how a TAG G can be transformed into a regular grammar (or equivalently into a finite-state automaton (FA)) that defines a (regular) superset of $\mathcal{L}(G)$. In the previous Section, we saw how to build a grammar G_{cf} that defines a CF superset of $\mathcal{L}(G)$. Therefore, we now only have to see how to define a regular superset of a CFL.

Finding a regular superset of a CFL is an old problem which has many theoretical solutions. However, as soon as "practical" constraints are added, such as "large" CFG and a not too "distant" superset, the solutions become much fewer. We will use the solution of [4] where it is shown how to transform a CFG into a deterministic FA called a set automaton (SA). A state of an SA is a pair $[I, J]$, in which I is a set of items (dotted productions of the form $A \rightarrow \alpha.\beta$) called the *control* set and J is a set of items called the *active* set. For a given string $w = a_1 \dots a_n$, a *configuration* of (the interpreter of) an SA $\mathcal{A} = (Q, T, \delta, q_0, F)$ is an element of $Q \times T^*$. The initial configuration c_0 is $c_0 = (q_0, w)$. A binary relation between configurations noted $\underset{\mathcal{A}}{\vdash}$ is defined by $(q, tx) \underset{\mathcal{A}}{\vdash} (q', x)$, iff $\delta(q, t) = q'$. The string w is recognized by $\mathcal{A}$ iff we have $c_0 \underset{\mathcal{A}}{\vdash} c_1 \dots \underset{\mathcal{A}}{\vdash} c_n$, with $c_n = (q_n, \varepsilon)$ and q_n is a final state in F.

Note that this SA, which could be extremely large, is not (pre-)constructed, but only the subpart selected by a given input $w = a_1 \dots a_n$ is constructed at run-time: that is the $n + 1$ states $q_0, q_1, \dots, q_n$ needed in configurations $c_0, c_1, \dots, c_n$.

4 Regular-Based Supertagging

However, the previous FA technology itself cannot be used as such, since only recognizing an input is not the purpose of a supertagger. Thus, we must see whether an SA can be transformed into a finite-state transducer (FT) whose outputs are supertags. Fortunately, SAs have the following property: for each configuration $c_i = ([I_i, J_i], a_{i+1} \dots a_n)$, $1 \leq i \leq n$, we are sure that in configuration c_{i-1}, the active set J_{i-1} contains items of the form $A \rightarrow \alpha.a_i\beta$. In other words, we can say that if a transition on some terminal t occurs between two configurations c_{i-1} and c_i, we have $a_i = t$ and, moreover, for each item of the form $A \rightarrow \alpha t.\beta$ in the active set of c_i, the CF production $A \rightarrow \alpha t\beta$ is anchored on a_i. That is, the production $A \rightarrow \alpha t\beta$ is a "supertag" w.r.t. G_{cf} for the i^{th} word of w.

Therefore, to summarize, if G is an LTAG, G_{cf} is its associated CFG as defined in Section 2 and if $\mathcal{A}$ is the SA associated with G_{cf} as defined in Section 3, the FT based on $\mathcal{A}$ which for each input word a_i outputs a set of CF productions is a "supertagger" for G_{cf}. In order to get a supertagger for the original LTAG G, we only have to exploit the reverse mapping which records for each production r of G_{cf}, the set of elementary trees τ of G from which r has been generated. If τ is an initial tree, there is a one-to-one mapping between τ and its associated (lexicalized) production, say r. Thus, if r is a CF supertag for some word a_i, we conclude that τ is a (TAG) supertag for a_i. If τ is an auxiliary tree, it generates two productions, say r_L and r_R. The auxiliary tree τ can be part of some complete parse for G only if both r_L and r_R are themselves part of some complete parse for G_{cf} (for the same input w). With auxiliary trees, two cases may arise. First, if both r_L and r_R are lexicalized productions and if r_L is a CF supertag for some a_i and if r_R is a CF supertag for some a_j, we may conclude that τ is a supertag for both a_i and a_j. Second, if only one of r_L and r_R is a lexicalized production,

say r_L, and if r_L is a CF supertag for some a_i, we may conclude that τ is a supertag for a_i. However, it is possible to take into account the fact that the (non-lexicalized) production r_R must have been "recognized" in the FT to validate τ as a supertag for a_i. Therefore, the FT is extended as follows. It also looks for complete items $A \rightarrow \alpha.$ of non-lexicalized productions $A \rightarrow \alpha$, $\alpha \in N^*$, and it records all these productions in a set R. Now, we say that τ is a (TAG) supertag for a_i only if r_L is a CF supertag for a_i and if r_R is in R. The (TAG) supertagger designed in such a way is called $\mathcal{S}_{\mathrm{SA}}$ in the experiment section 7.

We may use the mirror SA $\widetilde{\mathcal{A}}$ to increase the precision of this FT (See [4]). A mirror SA is an SA which uses the mirror grammar $\widetilde{G_{\mathrm{cf}}}$ which defines the mirror language of $\mathcal{L}(G_{\mathrm{cf}})$ (i.e., $\{a_n \ldots a_1 \mid a_1 \ldots a_n \in \mathcal{L}(G_{\mathrm{cf}})\}$). Of course, an interpreter of a mirror SA scans its input w from right to left. Therefore, such an FT proceeds in two passes, the first pass interprets w from left to right with $\mathcal{A}$, while the second pass interprets w from right to left with $\widetilde{\mathcal{A}}$. A production r of G_{cf} is a CF supertag for some word a_i, iff it is a CF supertag for a_i w.r.t. both $\mathcal{A}$ and $\widetilde{\mathcal{A}}$. This supertagger, based on set automata, which scans its input from left to right and from right to left is called $\mathcal{S}_{\mathrm{LRSA}}$ in the experiment section 7.

5 CF-Based Supertagging

In the previous section, in order to get a regular superset of the initial TAL, we built an intermediate CFG G_{cf} which defines a CF superset of this TAL. In this section we will see whether a general CF parser based on G_{cf} can be changed into a supertagger. Of course, doing that, on the one hand, we could expect a greater precision than the regular-based approach, but, on the other hand, we renounce the linear time behaviour for, at worst, a cubic time. However, since supertaggers typically handle short inputs (say sentences of a few dozen words), it could happen that the cubic run-time remains tractable. Since it exhibits good performances, we decided to use the guided Earley recognizer of [4] as a push-down transducer (PDT). As for the previous FTs, these PDTs will output for each input word position i a set of CF productions in G_{cf} which have to be mapped back onto the original LTAG G, as in the FT case, in order to get our CF-parsing based supertaggers. Note that we have decided to use both the SA $\mathcal{A}$ of G_{cf} and its mirror SA $\widetilde{\mathcal{A}}$ as a guide for the Earley recognizer. This means that we are sure that our CF-based supertaggers will at least exhibit the precision of $\mathcal{S}_{\mathrm{LRSA}}$.

If we consider the recognizer part of an Earley strategy, or more precisely its *Scanner* phase, we know that if an Earley item of the form $[A \rightarrow \alpha.t\beta, i]$ is in some table T_{j-1} and if the input $w = a_1 \ldots a_n$ is such that $a_j = t$, then the Earley item $[A \rightarrow \alpha t.\beta, i]$ is put in the next table T_j. In other words, this means that the production $A \rightarrow \alpha t\beta$ is a candidate CF supertag for the word a_j. However, in doing that, we are not sure that the complete item $[A \rightarrow \alpha t\beta., i]$ will be eventually an element of some table T_k, $k \geq j$. That is, no prefix of $a_{j+1} \ldots a_n$ is an element of the set of strings derived from β in G_{cf}. Of course, such a premature supertag detection may spoil the precision. Thus we decide to postpone any association between an anchor $a_j = t$ and a supertag such as $A \rightarrow \alpha t\beta$ until the complete recognition of its RHS $\alpha t\beta$.

This supertagger, which is (almost) only an Earley recognizer, is called $\mathcal{S}_{\mathrm{REC}}$ in the experiment described in Section 7.

We can also remark that, even if the recognition of some lexicalized production $A \rightarrow \alpha t\beta$ has been completed in a table T_k (i.e., $\exists [A \rightarrow \alpha t\beta., i] \in T_k$, $i < k$), this does not mean that

the corresponding sub-tree, say τ, (i.e., the subtree rooted at A whose daughter nodes are $\alpha t \beta$ and which spans between the indices i and k of the input) will necessarily be a part of some complete parse tree. Thus, we have built a second version of the CF supertagger, called $\mathcal{S}_{\text{RED}}$, in which we postpone the association between an anchor $a_j = t$ and a supertag $A \to \alpha t \beta$ until we are sure that a sub-tree such as τ is a part of a complete parse tree.

6 Precision of a Supertagger

If G is an LTAG, we note $\mathcal{G}$ its associated TAG parser. We assume that, by definition, the output of a supertagger $\mathcal{S}$ on a string $w = a_1 \dots a_n$ is a sequence of n sets. Each set, noted $\mathcal{S}_w^i$, $1 \leq i \leq n$ is the set of supertags anchored on a_i. Analogously, for G and a sentence $w = a_1 \dots a_n$, we note $\mathcal{G}_w^i$ the set of supertags anchored on the word a_i in the complete parse forest for w. Of course, no supertagger could do a more accurate job than a TAG parser and the sets $\mathcal{G}_w^i$ are taken as gold standard. By definition, the *precision score* (precision for short) of $\mathcal{S}$ on the word a_i of some w is the quotient $\frac{|\mathcal{G}_w^i|}{|\mathcal{S}_w^i|}$, the precision of $\mathcal{S}$ on a sentence w is the quotient $\frac{\sum_{1 \leq i \leq n} |\mathcal{G}_w^i|}{\sum_{1 \leq i \leq n} |\mathcal{S}_w^i|}$.[7] For a non-empty test set $\mathcal{T}$, the *average precision score for* $\mathcal{T}$ is the number $\frac{\sum_{w \in \mathcal{T}} \sum_{1 \leq i \leq |w|} |\mathcal{G}_w^i|}{\sum_{w \in \mathcal{T}} \sum_{1 \leq i \leq |w|} |\mathcal{S}_w^i|}$.

Moreover, as a consequence of the strictness of our approach, the *recall score* of all our supertaggers is 100%: no supertag which is a part of a complete parse (w.r.t. $\mathcal{G}$) is left out.

7 Test Material and Experiment Results

The measures presented in this section have been gathered on a 1.2GHz AMD Athlon PC running Linux. All processors are written in C and have been compiled with gcc without any optimization flag.

$\lvert N\rvert$	$\lvert T\rvert$	$\lvert P\rvert$	$\lvert G_{\text{cf}}\rvert$	max RHS #nt
33	476	1 131	17 129	26

Table 1: $G_{\text{cf}} = (N, T, P, S)$ facts

Our experiment is based upon a wide-coverage English grammar designed for the XTAG system [11]. This grammar consists of 1 132 tree templates that can be anchored on 476 anchors. As explained in Section 2, the XTAG grammar is first transformed into a CFG G_{cf}. In order to give an idea of the complexity of this grammar, the penultimate column of Table 1 gives the size of G_{cf} ($|G_{\text{cf}}| = \sum_{A \to \alpha \in P} |A\alpha|$), while the last column gives the number of nonterminal symbols that occur in the RHS of the longest production.

For our test set,[8] we have used sentences extracted from the Wall-Street Journal. We used 42 253 sentences for a total length of 1M words. An input word (inflected form) selects one or, more often, several terminal symbols (anchors). In this test set, a word selects about 11 anchors on average. The association between a word and its anchors is performed by means of a dictionary search. If an input word is not in the dictionary, we assign a default value to this

[7] Note that multi-anchored supertags are counted several times in $\mathcal{G}_w^i$ and $\mathcal{S}_w^i$.

[8] We use exactly the same test set as [4].

	Lexer	$\mathcal{S}_{\text{SA}}$	$\mathcal{S}_{\text{LRSA}}$	$\mathcal{S}_{\text{REC}}$	$\mathcal{S}_{\text{RED}}$	$\mathcal{G}$
#supertags	65.89	27.30	20.92	20.66	19.10	18.41
Precision	27.94%	67.43%	87.99%	89.10%	96.39%	100%
Recall	100%	100%	100%	100%	100%	100%

Table 2: Average number of supertags, average precision and average recall per word

unknown word: we assume that any unknown word is a noun and it thus selects the anchors associated with a noun. Of 1M words, 68 407 are unknown. Each anchor, in turn, selects several supertags (tree templates in G). On average, a word selects about 66 supertags. This initial selection process is performed by a *lexer*. Of course, a lexer may itself be seen as a supertagger, its performances are reported in Table 2 in the first column..

However, of these 42 253 sentences, 964 are extragrammatical w.r.t. G_{cf}, and 715 others are extragrammatical w.r.t. G,[9] thus, 1 679 input strings were left out of the actual test set leaving 40 574 sentences for a total of 925 605 words. In this test set, the lengths of individual sentences show great variations: with an average length of almost 23 words per sentence, there are single word sentences while the longest ones contain 97 words. The distribution of our test set is shown in Figure 2.

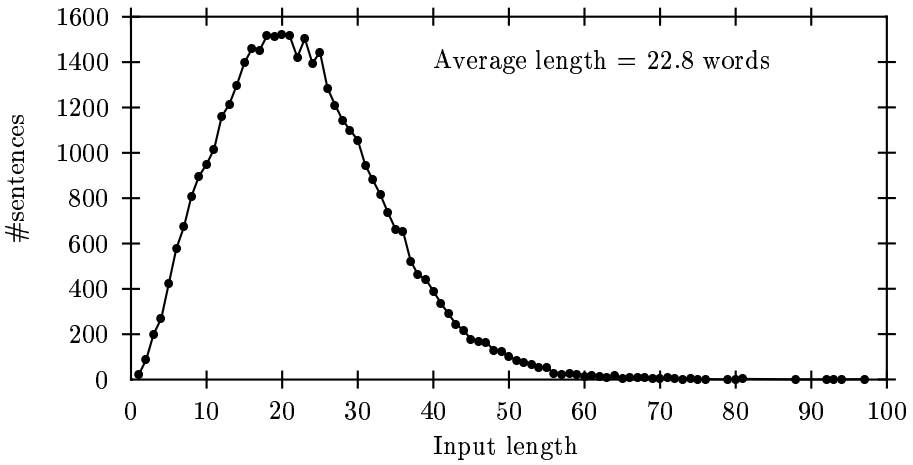


Figure 2: Distribution of the number of sentences by their lengths

The performances of our supertaggers are displayed in Table 2 while their precisions and run-times as a function of the input length are plotted in Figures 3 and 4 respectively.

In Figure 3, we note that the precision of each of our supertaggers seems to be largely independent of the sentence length, at least beyond a length of say 20 words. The high precision of $\mathcal{S}_{\text{RED}}$ can be interpreted as follows: on the one hand, the English XTAG definition is close to being context-free and, on the other hand, our TAG to CFG transformation, together with its PDT interpretation are accurate enough to allow this closeness to be shown. Lastly, we can remark that the savings of $\mathcal{S}_{\text{REC}}$ w.r.t. $\mathcal{S}_{\text{LRSA}}$ are rather limited.

In Figure 4, as expected, we note that both $\mathcal{S}_{\text{SA}}$ and $\mathcal{S}_{\text{LRSA}}$ seems to exhibit a linear run-time behaviour, while $\mathcal{S}_{\text{REC}}$ and $\mathcal{S}_{\text{RED}}$ show a super-linear behaviour (which should be cubic

[9]Note that some of these input strings are considered to be "extragrammatical" only because they exhausted the memory resource available for that test. As an example, this happened for a 158 word sentence!

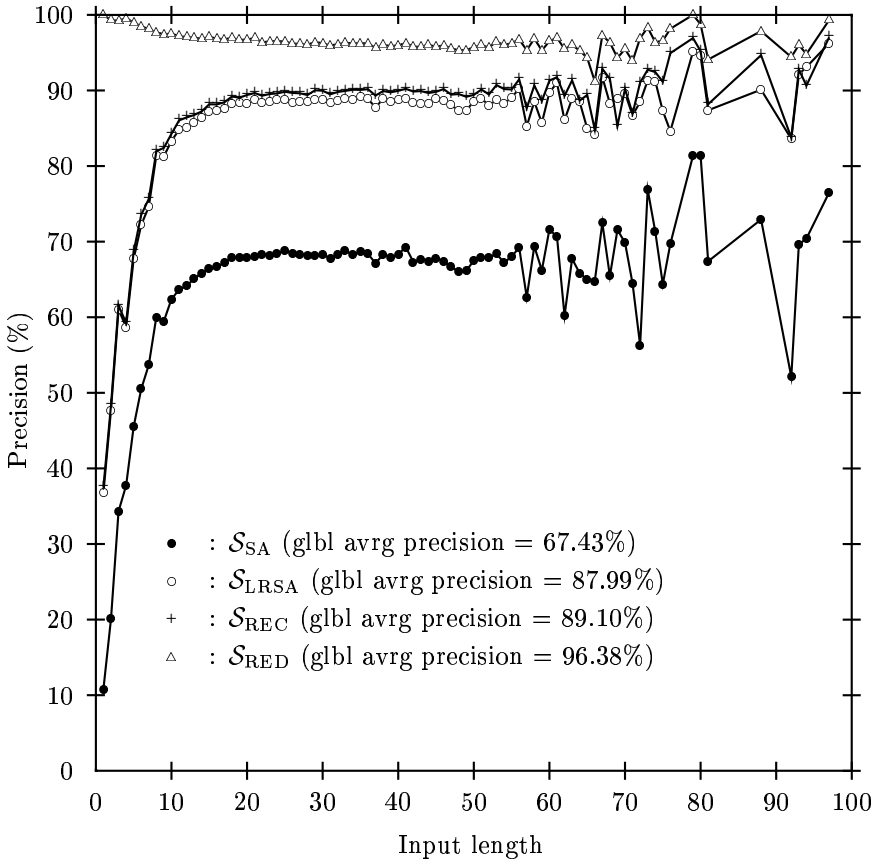


Figure 3: Supertaggers average precision score

at worst).

If we have to choose among these supertaggers, undoubtedly two candidates will emerge, $\mathcal{S}_{\mathrm{LRSA}}$ for its good trade-off between speed and precision and $\mathcal{S}_{\mathrm{RED}}$ for its high precision (if speed is not at a premium).

8 Conclusion

It is difficult to compare our results with previously published measures both in terms of run-times and in terms of performance.

The practical run-times of other supertaggers are almost never addressed, though we may assume that, at least theoretically, they run in linear time. In this paper, we have presented two classes of supertaggers. In the first class, the two supertaggers $\mathcal{S}_{\mathrm{SA}}$ and $\mathcal{S}_{\mathrm{LRSA}}$ run in linear time with average run-times of, respectively, 12ms and 18ms per sentence on our test set. In the second class, the two supertaggers $\mathcal{S}_{\mathrm{REC}}$ and $\mathcal{S}_{\mathrm{RED}}$ run in cubic time at worst. However, on typical small sentences that one usually handled in NLP, their run-times stay practicable: on average, $\mathcal{S}_{\mathrm{REC}}$ and $\mathcal{S}_{\mathrm{RED}}$ run respectively in 47ms and 73ms per sentence on our test set.

As concerns performance, the comparison of usual supertaggers with our work is also difficult since their goals are different. Classical supertaggers try to assign the *best* supertag to each word,[10] whereas we have designed a method in which each word selects a set of supertags which

[10] Or a small set of supertags as in [5].

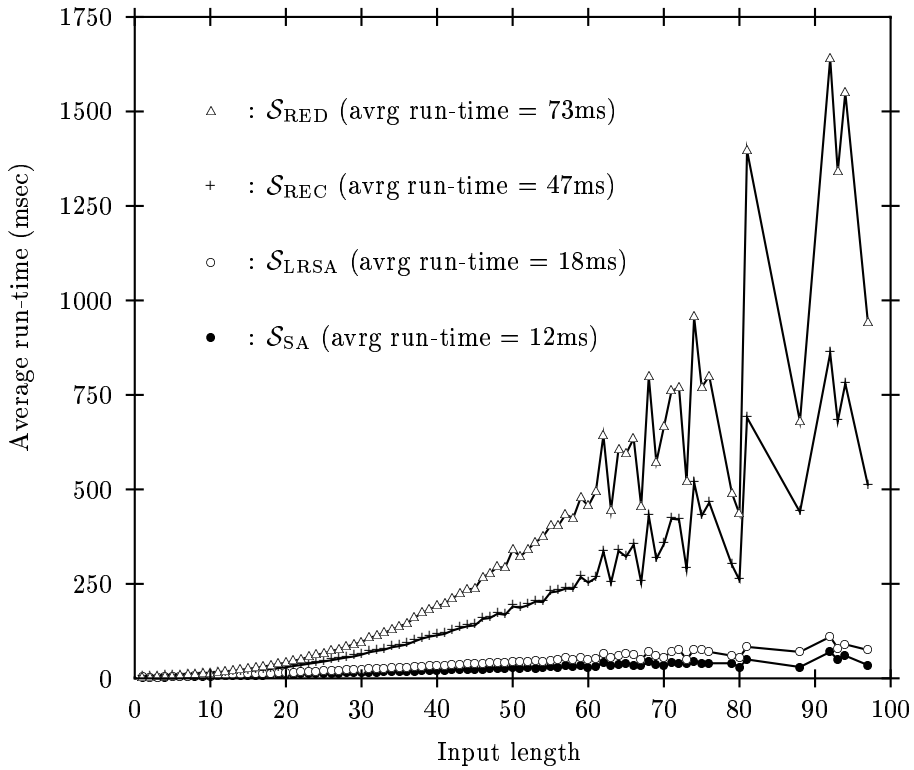


Figure 4: Supertaggers average run-times

does not exclude any supertag that will eventually be used in a valid complete derivation (i.e., we have designed a method whose recall score is 100%). In order to perform our precision score measures on a real-size application (both on a large scale grammar and on numerous unrestricted length sentences), we must have at our disposal a complete TAG parser to play the role of "gold supertagger". It seems that the only published measures which use the results of a TAG parser put considerable limits on the length of their input sentences.[11] These limits are due to the fact that the available TAG parsers, when working with a large grammar, are unable to process long sentences. To our knowledge, the only TAG parser which is capable of handling (almost) unrestricted length sentences has been described in [1]. It has been slightly modified and transformed into a gold supertagger which allows us to supertag 1M words of the WSJ (more than 42 000 WSJ sentences, up to a length of almost 100 words). As already remarked,[12] the usage of TAG derivations to perform some evaluations is more severe than the usage of annotated corpora. Nevertheless, we have reached very good precision scores: 88% for the linear time supertagger $\mathcal{S}_{\mathrm{LRSA}}$ and more than 96% for $\mathcal{S}_{\mathrm{RED}}$, a cubic time supertagger.

Moreover, in this conclusion, we want to stress that our supertagging methods are automatically deduced from the (TAG) grammar and do not need any training data. Our purpose was to build correctness-preserving supertaggers by using only structural information. We can note that such a correctness cannot be reached with statistical supertaggers, even if they select the top n supertags, for any value of n. In contrast to other approaches, our supertaggers may

[11] In [9], the reported experiment uses the English XTAG parser on 1350 WSJ sentences, the lengths of which are less than 16.

[12] From [9], we quote that "*... evaluation against LTAG derivation trees is much more strict and hence more significant than the crossing bracket precision and recall figures measured against skeletally bracketed corpora*".

be safely used as neutral (no useful information is lost) filters for other processors. These processors may well be classical statistical supertaggers or TAG parsers. Incidently, the complete TAG parser that we used as a gold supertagger in our experiments uses $\mathcal{S}_{\mathrm{RED}}$ as a filter.[13]

References

[1] François Barthélemy, Pierre Boullier, Philippe Deschamp, and Éric de la Clergerie. Guided parsing of range concatenation languages. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL'01)*, pages 42–49, University of Toulouse, France, July 2001.

[2] Pierre Boullier. A cubic time extension of context-free grammars. *Grammars*, 3(2/3):111–131, 2000.

[3] Pierre Boullier. On TAG parsing. *Traitement Automatique des Langues (T.A.L.)*, 41(3):759–793, 2000. Issued June 2001.

[4] Pierre Boullier. Guided Earley parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT 2003)*, Nancy, France, April 2003.

[5] John Chen, Srinivas Bangalore, and K. Vijay-Shanker. New models for improving supertag disambiguation. In *Proceedings of the 9th Conference of the European Chapter of the Association for Computational Linguistics (EACL'99)*, pages 188–195, Bergen, Norway, June 1999.

[6] Aravind K. Joshi. An introduction to tree adjoining grammars. In A. Manaster-Ramer, editor, *Mathematics of Language*, pages 87–114. John Benjamins, Amsterdam, 1987.

[7] Aravind K. Joshi and Bangalore Srinivas. Disambiguation of super parts of speech (or supertags): Almost parsing. In *Proceedings of the 17th International Conference on Computational Linguistics (COLING'94)*, Kyoto, Japan, 1994.

[8] Yves Schabes, Anne Abeillé, and Aravind K. Joshi. Parsing strategies with 'lexicalized' grammars: Application to tree adjoining grammars. In *Proceedings of the 12th International Conference on Computational Linguistics (COLING'88)*, Budapest, Hungary, 1988.

[9] Bangalore Srinivas. *Advances in Probabilistic and Other Parsing Technologies*, volume 16 of *Text, Speech and Language Technology*, chapter Performance Evaluation of Supertagging for Partial Parsing, pages 203–220. Kluwer Academic Publishers, H. Bunt and A. Nijholt edition, 2000.

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

[11] The research group XTAG. A lexicalized tree adjoining grammar for English. Technical Report IRCS 95-03, Institute for Research in Cognitive Science, University of Pennsylvania, Philadelphia, PA, USA, March 1995.

[13] In fact $\mathcal{S}_{\mathrm{RED}}$ is used as the guiding phase of our (guided) TAG parser (See [1] and [4] for a notion of guided parsing). In a further paper, we plan to relate the benefits of such a preprocessing for a full TAG parser.