Efficient HPSG Parsing with Supertagging and CFG-filtering

Takuya Matsuzaki [1] **Yusuke Miyao** [1] **Jun'ichi Tsujii** [1,2,3]
1. Department of Computer Science, University of Tokyo
Hongo 7-3-1, Bunkyo-ku, Tokyo 113-0033, Japan
2. School of Computer Science, University of Manchester
3. NaCTeM (National Center for Text Mining)
{matuzaki, yusuke, tsujii}@is.s.u-tokyo.ac.jp

Abstract

An efficient parsing technique for HPSG is presented. Recent research has shown that supertagging is a key technology to improve both the speed and accuracy of lexicalized grammar parsing. We show that further speed-up is possible by eliminating non-parsable lexical entry sequences from the output of the supertagger. The parsability of the lexical entry sequences is tested by a technique called CFG-filtering, where a CFG that approximates the HPSG is used to test it. Those lexical entry sequences that passed through the CFG-filter are combined into parse trees by using a simple shift-reduce parsing algorithm, in which structural ambiguities are resolved using a classifier and all the syntactic constraints represented in the original grammar are checked. Experimental results show that our system gives comparable accuracy with a speed-up by a factor of six (30 msec/sentence) compared with the best published result using the same grammar.

1 Introduction

Deep syntactic analysis by lexicalized grammar parsing offers a solid basis for intelligent text processing, such as question-answering, text mining, and machine translation. In those tasks, however, a large amount of text has to be processed to build a useful system. One of the main difficulties of using lexicalized grammars for such large-scale applications is the inefficiency of parsing caused by complicated data structures used in such grammars.

Recent research showed the importance of supertagging for the speed, as well as the accuracy, of lexicalized grammar parsing [Clark and Curran, 2004; Ninomiya *et al.*, 2006]. Supertagging is a tagging process where lexical entries are assigned to the words in the input sentence [Bangalore and Joshi, 1999]. In lexicalized grammars, a lexical entry encodes many constraints on how a word is combined with surrounding words or phrases. The parsing efficiency is therefore increased by supertagging because it makes the search space explored by the parser much narrower.

The current accuracy of the supertagger is, in general, not sufficient to use it as a single-tagger, which assigns only one lexical entry for each word, because the number of supertags is generally large (more than 1000 in our case) and only one or two tagging errors in a sentence will cause a parse failure in many cases. In previous research [Clark and Curran, 2004; Ninomiya *et al.*, 2006], the problem is overcome by assigning several supertags to each word in a sentence, i.e., multi-supertagging; the supertagger initially assigns only a small number of lexical entries to each word and the number of lexical entries is gradually increased until the parser finds a successful parse. In short, the parser 'takes over' a part of the supertagger's task and resolves a certain amount of the lexical ambiguity by itself to avoid parse failures.

In this paper, we show that, by making the supertagger more powerful, the workload of the parser can be further reduced and the overall system becomes more efficient. Specifically, we combine the supertagger with a CFG that approximates the original lexicalized grammar, to enumerate *maybe-parsable* supertag assignments in the order of their scores given by the supertagger. We say a supertag sequence is maybe-parsable if the sequence is parsable by the approximating CFG. The most time-consuming part of the enumeration algorithm is parsing of the input sentence with the approximating CFG. However, we can do this CFG parsing efficiently because the CFG is generally sparse, i.e., the combination of symbols that appear in a CF-rule is highly restricted.

The enumerated supertag sequences are parsed by an HPSG parser one by one, until a successful parse is obtained. Though the enumerated supertag sequences are not necessarily parsable by the original HPSG, we observed in the experiments that the parser finds a successful parse within only a few maybe-parsable supertag sequences for most sentences.

The biggest advantage of our approach is that the HPSG parser used in the last stage of the system, which is the most computationally demanding part in the previous approach, can now be replaced by a simpler and more efficient parsing algorithm. Our HPSG parser is implemented as a classifier-based, deterministic shift-reduce parser that is guided by a CFG-forest. The CFG-forest is created by the approximating CFG, and it approximates the HPSG-forest that would be obtained if the input supertag sequence were fully parsed with the HPSG. We do not need to keep many hypothetical sub-analyses represented by complex feature structures, as in the chart parsers used in the previous approach, since the input to the HPSG parser is single-supertagged and also we can know almost surely which sub-analysis grows into a well-formed parse tree by referring to the CFG-forest.

2 Background

2.1 Head-driven Phrase Structure Grammar

HPSG [Pollard and Sag, 1994] is a linguistic theory based on the lexicalized grammar formalism. An instance of HPSG grammar mainly consists of two parts: a small number of rule schemata and a large number of lexical entries. The rule schemata represent general grammatical constraints, while the lexical entries in the lexicon express word-specific characteristics. In HPSG, both lexical entries and phrasal constituents are represented by typed feature structures called *signs* and applications of the rule schemata are implemented by *unification* of signs and schemata.

Figure 1 presents an example of HPSG parsing for the sentence "*I like it*."[1] First, three lexical entries are selected from ones associated with each word in the lexicon. Then, the lexical entries of "*like*" and "*it*" are combined by applying the Head-Complement schema to them and then the resultant phrasal sign of the verb phrase "*like it*" is combined with the lexical entry of "*I*" by the Subject-Head schema.

Variations of syntactic constructions allowed for a word are represented by different lexical entries associated to the word in the lexicon. These variations include not only ones with different subcategorization frames (e.g., transitive and intransitive) but also 'transformational' variations such as passivization and wh-extraction. The form of a parse tree constructed upon a word sequence is hence highly constrained once a lexical entry is selected for each word.

2.2 Supertagging

Supertagging is a process where the words in the input sentence are tagged with 'supertags.' In our case, a supertag is a *lexical template*, which is a common structure shared among lexical entries for different words.

Supertagging can be formulated as a sequence labeling task and several types of techniques have been applied to it. We selected a simple approach proposed by Clark [2002], which uses a maximum entropy classifier. The conditional probability of a supertag sequence $\mathbf{t} = t_1 \dots t_n$ given a (POS-tagged) sentence $\mathbf{s}$ is calculated as:

$$P(\mathbf{t}|\mathbf{s}) = \prod_{i=1}^{n} P(t_i|\mathbf{s}) = \prod_{i=1}^{n} \frac{1}{Z_i} \exp\left(\mathbf{W} \cdot \Phi(\mathbf{s}, t_i, i)\right) \quad (1)$$

where $\mathbf{W}$ is a vector of feature weights, $\Phi(\mathbf{s}, t_i, i)$ is the feature vector, and Z_i is a normalization constant.

As is clear from the equation above, the supertagger neglects the parsability of the supertag sequence. We empirically show that, by combining a CFG-filter with the supertagger, it is possible to enumerate supertag sequences which are most of the case parsable, in the order of the probability defined in Eq.(1), with only a small additional processing time.

[1]Feature structures used in the example are very simplified ones; a much more complicated grammar is used in the experiments.

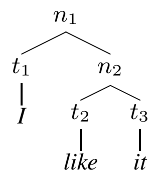

Figure 2: Analysis by the approximating CFG

2.3 CFG-filtering

CFG-filtering [Kiefer and Krieger, 2000; Torisawa *et al.*, 2000] is a parsing technique for unification-based grammars. In HPSG parsing based on CFG-filtering, an input sentence is first parsed by a CFG that approximates the original HPSG and then the CFG-parsing is 'replayed' using the HPSG. The CFG parsing in the first stage is much faster than normal HPSG parsing since the unification operations are replaced by identity checking of atomic symbols of CFG, which is a much more faster operation than unification.

The non-terminal symbols N, the terminal symbols T, and the set of CF-rules R in the approximating CFG respectively represent (abstracted) phrasal signs, (abstracted) lexical entries, and instantiations of the rule schemata. For example, the CFG tree in Figure 2, which is an analysis of the sentence "*I like it*." by an approximating CFG, corresponds to the HPSG parse tree in Figure 1. In Figure 2, n_1, n_2, t_1, t_2, and t_3 are non-terminal and terminal symbols of the approximating CFG which represent feature structures at the corresponding positions in the HPSG parse tree.

The number of possible phrasal signs in an HPSG is infinite in general. To approximate the HPSG by a CFG, we thus need to abstract away several parts of the phrasal signs to keep the number of abstracted signs finite. The abstraction of the feature structures is specified by means of a restrictor [Shieber, 1985] on the feature structures; a restrictor takes a feature structure and overwrites several feature values in it to the most general values allowed for the features.

We use one of the algorithms proposed by Kiefer and Krieger [2000] to approximate the HPSG. The algorithm finds all possible abstracted phrasal signs by iteratively applying the schemata to abstracted signs that have been found so far. A sketch of the algorithm is as follows:

```
1:  Restrict all the lexical entries and obtain the set of terminal symbols $T$
2:  $i \leftarrow 0; N_i \leftarrow T$
3:  **while** $N_i \neq N_{i+1}$ **do**
4:      $N_{i+1} \leftarrow N_i$
5:      **for all** $n_1, n_2 \in N_i$ **do**
6:          **for all** schema $s$ **do**
7:              Apply $s$ to $n_1$ and $n_2$ as daughters and obtain the mother phrasal sign $m$
8:              Restrict $m$ to $m'$
9:              **if** $m' \notin N_{i+1}$ **then**
10:                 $N_{i+1} \leftarrow N_{i+1} \cup \{m'\}$
11:                 $R \leftarrow R \cup \{m' \rightarrow n_1 n_2\}$
12: $N \leftarrow N_i$
```

The restrictor defines a many-to-one mapping from the set of supertags (i.e., lexical entries) to the set of terminal sym-

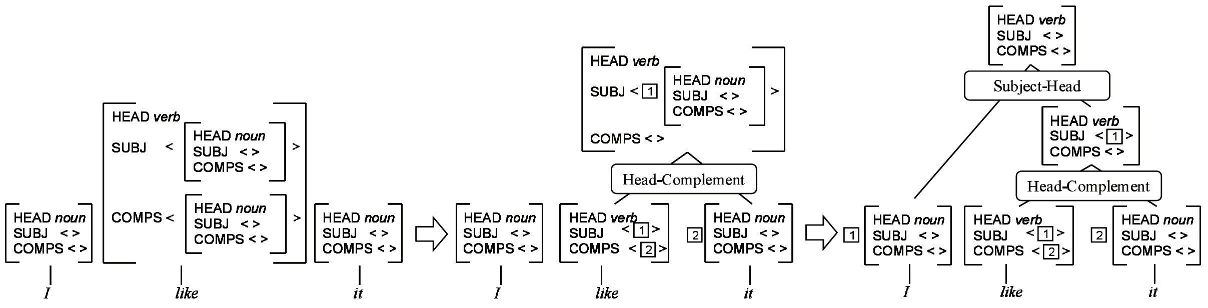
HEAD noun
SUBJ < >
COMPS < >
HEAD verb
SUBJ < HEAD noun SUBJ < > COMPS < > >
COMPS < HEAD noun SUBJ < > COMPS < > >
I
like
it
Head-Complement
Subject-Head

Figure 1: HPSG parsing

bols T. We however pretend there is a one-to-one mapping between these two sets by allowing multiple occurrences of the same terminal symbols in T, if necessary, and remembering which supertag was mapped to which terminal symbol. In the rest of the paper, we assume such a one-to-one relation and treat terminal symbols and supertags interchangeably.

An important property of the CFG approximation is that the language of the obtained CFG is a superset of the set of parsable supertag sequences. In other words, if a supertag sequence is not parsable by the CFG, it is also not parsable by the HPSG. We use this property to eliminate non-parsable supertag sequences from the output of the supertagger.

3 An efficient multi-stage parsing algorithm for HPSG

Our parsing system takes as input a POS-tagged sentence, $(\langle w_1, p_1\rangle, \ldots, \langle w_n, p_n\rangle)$ where w_i is a word and p_i is a POS tag, and outputs an HPSG parse tree. The algorithm is as follows:

```
1: for all $\langle w_i, p_i\rangle$ in the input sentence do
2:     Assign scores by the supertagger to supertags associated to $w_i$ in the lexicon
3: $j \leftarrow 0$
4: repeat
5:     $j \leftarrow j + 1$
6:     $\mathbf{t} \leftarrow j$-th best maybe-parsable supertag sequence
7:     Parse $\mathbf{t}$ with the approximating CFG
8:     Select an HPSG parse tree by the disambiguator, using the CFG forest made in the above step
9: until a well-formed HPSG parse tree is obtained
10: return the selected HPSG parse tree
```

The approximating CFG is generally not equivalent to the HPSG, and therefore, a maybe-parsable supertag sequence might not be parsable by the HPSG. In such cases, the disambiguator at line 8 fails to find a well-formed HPSG parse tree and another supertag sequence is tried in the next iteration. The disambiguator might fail even when the maybe-parsable sequence is truly parsable by the HPSG because the CFG-forest created at line 7 may contain a tree not licensed by the HPSG and the disambiguator might mistakenly try to reproduce such a tree with the HPSG. However, we found such cases are empirically rare and thus decided to simply try the next supertag sequence in such cases. The rest of this section describes the algorithm of enumerating maybe-parsable supertag sequences (line 6) and the disambiguator (line 8).

3.1 Enumeration of maybe-parsable supertag sequences

The algorithm is based on Jiménez and Marzal's algorithm [Jiménez and Marzal, 2000], which enumerates, given a weighted CFG and an sentence, the N-best parse trees for the sentence in order of their weights. In their algorithm, the input sentence is first parsed with the weighted CFG using the CKY algorithm and then the N-best parse trees are obtained by enumerating the top N derivations of the root edge, i.e., the edge which spans the whole input and whose label is the start symbol. The N-best derivations of an edge e are recursively obtained by combining the N-best sub-derivations of the sub-edges of e and selecting the top N derivations from the combinations. In reality, it is not necessary to calculate all the N-best derivations for every edge in the chart because the N-best sub-derivations can be enumerated in a 'lazy' manner. See their paper for further details on their algorithm.

To obtain the N-best parsable supertag sequences, instead of the N-best parse trees, we modify Jiménez and Marzal's algorithm as follows. Our algorithm takes as input a sentence $\mathbf{w} = (w_1, w_2, \ldots)$ and a list of scored supertags for each word, $\{\langle t_{ij}, s_{ij}\rangle\}$, where t_{ij} is the j-th scored supertag for w_i and s_{ij} is the log-probability of t_{ij}. Given the input and the approximating CFG, G_a, we make a weighted CFG such that each CF-rule in G_a has a weight zero and each 'leaf rule' $t_{ij} \rightarrow w_i$ has a weight s_{ij}. Note that the score of a derivation defined by the weighted CFG equals the log-probability of the supertag sequence on the *fringe* of the derivation. The fringe of a derivation is the sequence of pre-terminals (i.e., t_{ij}s) of the tree of the derivation. In the first phase of the algorithm, we parse the input sentence with the weighted CFG by using the CKY algorithm. In the second phase of the algorithm, we enumerate the N-best fringes of the derivations of the root edge. Just as in Jiménez and Marzal's algorithm, we can recursively obtain the N-best fringes for an edge e by concatenating the N-best sub-fringes for the sub-edges of e and selecting the top N fringes from them in a lazy manner.

We made several modifications to the basic algorithm described above in order to make it more efficient. The first is that we replace the CKY algorithm used in the first step of the algorithm with an agenda-based best-first parsing algorithm. We further split the agenda-based parsing into two stages. The first stage of the best-first parsing stops when a root edge is found. In most cases, the first maybe-parsable supertag sequence is truly parsable with the original HPSG. In such cases, we need only the best-scored parse tree since the first maybe-parsable supertag sequence is its fringe. When the second supertag sequence is requested, we start the second stage, in which the best-first parsing is continued until all the edges scored greater than $(\alpha - \theta)$ are added to the chart, where α is the score of the best-scored parse tree and θ is a user-defined threshold. At the end of the second stage, any edge used in a parse tree with a score greater than $(\alpha - \theta)$ is stored in the chart and therefore we can find all the maybe-parsable sequences scored greater than $(\alpha - \theta)$ without fully parsing the input sentence.

Another modification is that we set the weight of the leaf rule $t_{ij} \rightarrow w_i$ to $(s_{ij} - s_{i1})$ instead of s_{ij}. Note that this modification does not alter the supertag sequences enumerated by the algorithm or their orders. By this modification, generation of edges with small scores is suppressed for the same reason as in A*-parsing [Klein and Manning, 2003].

The last modification is that, before the agenda-based parsing starts, we discard all the supertags for word w_i whose scores are less than $s_{i1} - \beta$, where β is a user-defined threshold. By this modification, we might lose some maybe-parsable sequences in which those discarded supertags appear. In the experiments, however, we found that the accuracy of parse trees created upon such supertag sequences with low scores is generally low. We therefore decided to use this thresholding with β for the efficiency of the enumeration.

3.2 Disambiguation by shift-reduce parsing with a guiding forest

We use an algorithm based on deterministic shift-reduce parsing to select an HPSG parse tree from those that can be constructed upon a given supertag sequence. Although we could use other disambiguation algorithms, the deterministic-parser-like algorithm is well suited to our framework because there is almost no need to keep multiple hypothetical sub-analyses (as in bottom-up chart parsing algorithms) to avoid search failure, since the supertag sequence given to the parser is maybe-parsable and we also have a CFG-forest on the supertag sequence, which guides the shift-reduce parser just like an LR-table with infinite look-ahead.

Deterministic, classifier-based approaches to disambiguation have been applied to dependency parsing (e.g., [Yamada and Matsumoto, 2003; Nivre and Scholz, 2004]) and CFG parsing (e.g., [Sagae and Lavie, 2005]). Note that we cannot apply these algorithms to HPSG parsing in a straightforward manner since they cause many parse failures for highly constrained grammars such as HPSG; to avoid parse failures, we need the approximating CFG for filtering the supertag sequences and for making the guiding CFG forest.

The disambiguator has two components, a stack of phrasal and lexical signs, S, and a queue of lexical signs (i.e., supertags), Q. The input to the disambiguator is a POS-tagged sentence, a maybe-parsable supertag assignment $\mathbf{t} = (t_1, t_2, \dots)$, and a forest, f, created by parsing $\mathbf{t}$ with the approximating CFG, using the CKY algorithm. The disambiguation algorithm is as follows:

```
1: $S \leftarrow$ empty stack; $Q \leftarrow (t_1, t_2, \dots)$
2: while $Q$ is not empty or length($S$) $> 1$ do
3:     if $\Gamma(S, f) = \phi$ then
4:         return fail
5:     $a \leftarrow \operatorname{argmax}_{a \in \Gamma(S,f)} \mathbf{W} \cdot \Phi(S, Q, a)$
6:     $\langle S, Q \rangle \leftarrow \mathrm{apply}(a, \langle S, Q \rangle)$
7: $r \leftarrow$ pop($S$)
8: return r
```

In the algorithm, a denotes a parser action, $\Gamma(S, f)$ is the set of possible parser actions given the stack S and the forest f, $\mathbf{W}$ is the vector of feature weights, and $\Phi(S, Q, a)$ is the feature vector. There are two types of parser action:

SHIFT: it pops a lexical sign from Q and pushes it into S.

APPLY_SCHEMA_X: it applies an n-ary schema X to the top n elements of S and replaces them with the resulting mother phrasal sign.

For example, the HPSG parse tree in Figure 1 is constructed by the following sequence of actions: SHIFT, SHIFT, SHIFT, APPLY_SCHEMA_Head_Complement, APPLY_SCHEMA_Subject_Head. The weight vector $\mathbf{W}$ is obtained with the averaged-perceptron algorithm [Collins and Duffy, 2002] with the polynomial kernel of degree 2.

The set of possible actions $\Gamma(S, f)$ is determined so that the parser never 'goes out of the forest.'; $\Gamma(S, f)$ is created by first mapping each element in S, i.e., signs of subtrees, to its corresponding node in f, then selecting actions which can lead to at least one complete CFG parse tree in f that includes all the nodes to which some stack elements are mapped, and finally eliminating from them any such APPLY_SCHEMA_Xs whose schema X is not applicable to the stack elements on the top of S.

4 Experiments

In this section, we first give a brief summary of the specific HPSG grammar used in the experiments and also describe the test/training data we used. We then give implementation details of the supertagger, the CFG-filter, and the disambiguator. Finally the experiments and their results are discussed.

4.1 Enju grammar

We used Enju version 2.1[2], an HPSG grammar for English [Miyao *et al.*, 2005]. The design of the grammar basically follows the definition of [Pollard and Sag, 1994]. Twelve schemata are defined in the grammar (9 binary schemata and 3 unary schemata). The lexicon of the grammar was extracted from Sections 02-21 of the Penn Treebank [Marcus *et al.*, 1994] (39,832 sentences). The grammar consists of 2,253 lexical entries for 9,567 words.

A program that converts the Penn Treebank into an HPSG treebank is also distributed with the grammar. We used it to make the training and test data for the experiment. A standard training/development/test split of the data is used; i.e., we

[2]http://www-tsujii.is.s.u-tokyo.ac.jp/enju/index.html

description	feature templates
surrounding words	w_{-1}, w_0, w_1
surrounding POS tags	$p_{-2}, p_{-1}, p_0, p_1, p_2, p_3$
combinations	$w_{-1}w_0, w_0w_1, p_{-1}w_0, p_0w_0, p_1w_0,$ $p_0p_1p_2p_3, p_{-2}p_{-1}p_0, p_{-1}p_0p_1,$ $p_0p_1p_2, p_{-2}p_{-1}, p_{-1}p_0, p_0p_1, p_1p_2$

Table 1: Features used in the supertagger

used the HPSG treebanks created from Section 02-21/22/23 of the Penn Treebank as training/development/test data.

4.2 Implementation details

HPSG supertagger

Table 1 lists the features used in our supertagger. In the table, p_x and w_x respectively denote a POS tag and a word at relative position x from the target word. We used the same feature set as one used in Ninomiya et al.'s HPSG supertagger so that the comparison of our system and theirs becomes more meaningful. The number of supertags is 2,253 (i.e., the number of lexical entries). [3]

CFG-filter

We created an approximating CFG from the Enju grammar by Kiefer and Krieger's algorithm. Examples of the features we restricted in the approximation are PHON (phonology feature), SYNSEM:LOCAL:CONT (semantic structures of the phrase), and SYNSEM:LOCAL:CAT:HEAD:AGR (agreement feature). The CFG contains 1,241 terminal symbols, 20,647 non-terminal symbols, and 458,988 rules.

Disambiguator

Table 2 lists features used in the disambiguator.[4] Features 1-4 are adaptations of the features used in Sagae and Lavie's CFG shift-reduce parser [Sagae and Lavie, 2005], and features 5-9 are adaptations of ones used in Miyao and Tsujii's CKY-style HPSG parser [Miyao and Tsujii, 2005]. Feature 10 includes several types of valency constraint read off from the phrasal/lexical signs. For example, from a lexical sign for a ditransitive usage of "give", we extract two features, "subject=NP" and "complement=NP_NP".

4.3 Experimental results

We evaluated the speed and the accuracy of the proposed method on sentences in the test data of ≤ 40 words (2,162 sentences) and ≤ 100 words (2,299 sentences). We measured the accuracy of the parse results by the precision (LP) and recall (LR) of the (labeled) predicate-argument relations output by the parser. See [Miyao and Tsujii, 2005] for details of the definition of the predicate-argument relations. The F_1 score

1. surface form of the head word of x
2. POS tag of the head word of x
3. lexical template of the head word of x
4. phrasal category of x (e.g., S, NP, NX)
1-4 are for $x \in \left\{ \begin{array}{c} S(0), \ldots, S(3), Q(0), \ldots, Q(3), \\ S(0).\text{left\_dep}, S(0).\text{right\_dep}, \\ S(1).\text{left\_dep}, S(1).\text{right\_dep} \end{array} \right\}$
5. distance between head words of $S(0)$ and $S(1)$
6. whether a comma exists between $S(0)$ and $S(1)$
7. whether a comma exists inside $S(0)$ or $S(1)$
8. pair of $S(1)$.rmost_pos and $S(0)$.lmost_pos
9. number of words dominated by $S(0)$ and $S(1)$
10. valence features of $S(0), S(1), Q(0)$, and $Q(1)$

Table 2: Features used in the parser

is the harmonic mean of LP and LR. All the timing information was collected on an AMD Opteron server with a 2.4-GHz CPU. The two parameters, β and θ, were manually tuned using the development set; $\beta = \log(1000)$ and $\theta = \log(100)$.

Table 3 lists the results of the parsing experiments on the test set. The table also lists several reported results on the same test set by other HPSG parsers with the same grammar: Ninomiya et al.'s parser is the model III in [Ninomiya *et al.*, 2006], which is the supertagger-based HPSG parser briefly explained in Section 1. Miyao and Tsujii's parser is a CKY-style HPSG parser [Miyao and Tsujii, 2005][5]. Both parsers use maximum entropy models for the disambiguation, and the main difference between them is the inclusion of a supertagger by Ninomiya et al.'s parser. The higher efficiency of our approach is clear from the table: out system runs around six times faster than Ninomiya et al.'s parser with comparable accuracy.

On the test set of the sentences ≤ 100 words, our method found a well-formed parse for 97.1% of the sentences.[6] In preliminary experiments on the development set, we found the rate of successfully parsed sentences reached nearly 100% when we chose larger values for β. However, for such settings, average parse time significantly increased and the F_1 score did not improve because while the recall slightly improved, the precision slightly deteriorated. For example, with $\beta = \log(10^5)$, 99.6% of the sentences in the development set got a parse with an average parse time of 52.9 ms. This fact means that when the first maybe-parsable supertag sequence is assigned very low probability by the supertagger, the enumeration algorithm needs to generate many edges until it finds the sequence, but the HPSG parse created on such a sequence is not so accurate.

Table 4 shows the cumulative percentages of the sentences on which the parser found a well-formed parse within a certain number of maybe-parsable supertag sequences. This experiment was done on the development set and with the same parameter values as the above experiment. For about 95% of the sentences, the first maybe-parsable supertag sequence

[3]Many of them do not, however, appear in the training treebank. The 'effective' size of the tag set is thus around 1,300 since the rest of supertags, which do not appear in the training data, are assigned very low probabilities by the supertagger and hence rarely used.

[4]$S(i)$ and $Q(i)$ denote i-th elements from the top of the stack and the queue. $S(n)$.left_dep (resp. $S(n)$.right_dep) denotes the most recently found lexical dependent of the head word of $S(n)$ that is to the left (resp. right) of $S(n)$'s head; $S(n)$.lmost_pos ($S(n)$.rmost_pos) denotes the POS tag of the left-most (right-most) word of $S(n)$.

[5]Enju parser (ver2.1). http://www-tsujii.is.s.u-tokyo.ac.jp/enju

[6]For those sentences on which the parser failed, we collected partial parse results by applying the disambiguator (without a guiding forest) on the 1-best supertag sequence and evaluated the predicate-argument relations identified in those partial parse results.

	Sentences $\leq$ 40 words				Sentences $\leq$ 100 words			
Parsers	LP	LR	F_1	Avg. Time	LP	LR	F_1	Avg. Time
this paper	87.15	86.65	86.90	25.9 ms	86.93	86.47	86.70	29.6 ms
Ninomiya et al. [2006]	87.66	86.53	87.09	155 ms	87.35	86.29	86.81	183 ms
Miyao and Tsujii [2005]	85.33	84.83	85.08	509 ms	84.96	84.25	84.60	674 ms

Table 3: Results of parsing on Section 23

# of invocations	1	$\leq$5	$\leq$10	$\leq$20	$\leq$111
cumulative%	94.89	96.90	97.14	97.32	97.63

Table 4: Number of enumerated supertag sequences until a successful parse

Sub-module	Avg. time (%)
POS tagging	7.4 ms (27%)
Supertagging	7.0 ms (26%)
Enumeration of supertag sequences	3.4 ms (12%)
CKY parsing of a supertag sequence	0.6 ms (2.3%)
Disambiguation by the HPSG parser	7.7 ms (28%)
other	1.9 ms
total	28 ms (100%)

Table 5: Breakdown of the average processing time

was truly parsable with the HPSG and more than five invocations of the HPSG parser were needed for only 0.7% of the sentences.[7] For any failed sentences, the enumerator returned *no* maybe-parsable supertag sequences, i.e, the CFG-parser failed on those sentences. These observations mean that the CFG approximation was a fairly tight one, and hence, the number of 'unfruitful' invocations of the HPSG parser, which do not give a well-formed parse tree, was kept small.

Table 5 shows a breakdown of the average processing time on the development set. The table suggests that our approach significantly reduced the 'workload' of the HPSG parser by its small overhead for the enumeration of maybe-parsable supertag sequences; the processing time used in the HPSG parser is now very close to that for POS tagging.

[7] 111 at the right-most column is the maximum number of maybe-parsable supertag sequences enumerated for a sentence.

References

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

[Clark and Curran, 2004] S. Clark and J. R. Curran. The importance of supertagging for wide-coverage ccg parsing. In *Proc. COLING*, pages 282–288, 2004.

[Clark, 2002] S. Clark. Supertagging for combinatory categorial grammar. In *Proc. TAG+6*, pages 19–24, 2002.

[Collins and Duffy, 2002] M. Collins and N. Duffy. New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In *Proc. ACL*, pages 263–270, 2002.

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

[Kiefer and Krieger, 2000] B. Kiefer and H.-U. Krieger. A context-free approximation of head-driven phrase structure grammar. In *Proc. IWPT*, pages 135–146, 2000.

[Klein and Manning, 2003] D. Klein and C. D. Manning. A* parsing: Fast exact viterbi parse selection. In *Proc. HLT-NAACL*, pages 40–47, 2003.

[Marcus *et al.*, 1994] M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz. Building a large annotated corpus of english: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1994.

[Miyao and Tsujii, 2005] Y. Miyao and J. Tsujii. Probabilistic disambiguation models for wide-coverage HPSG parsing. In *Proc. ACL*, pages 83–90, 2005.

[Miyao *et al.*, 2005] Y. Miyao, T. Ninomiya, and J. Tsujii. Corpus-oriented grammar development for acquiring a Head-driven Phrase Structure Grammar from the Penn Treebank. In *Natural Language Processing - IJCNLP 2004*, volume 3248 of *LNAI*, pages 684–693. Springer-Verlag, 2005.

[Ninomiya *et al.*, 2006] T. Ninomiya, T. Matsuzaki, Y. Miyao, Y. Tsuruoka, and J. Tsujii. Extremely lexicalized models for accurate and fast HPSG parsing. In *Proc. EMNLP*, pages 155–163, 2006.

[Nivre and Scholz, 2004] J. Nivre and M. Scholz. Deterministic dependency parsing of english text. In *Proc. COLING*, pages 64–70, 2004.

[Pollard and Sag, 1994] C. Pollard and I. A. Sag. *Head-Driven Phrase Structure Grammar*. University of Chicago Press, 1994.

[Sagae and Lavie, 2005] K. Sagae and A. Lavie. A classifier-based parser with linear run-time complexity. In *Proc. IWPT*, pages 125–132, 2005.

[Shieber, 1985] S. M. Shieber. Using restriction to extend parsing algorithms for complex-feature-based formalisms. In *Proc. ACL*, pages 145–152, 1985.

[Torisawa *et al.*, 2000] K. Torisawa, K. Nishida, Y. Miyao, and J. Tsujii. An HPSG parser with CFG filtering. *Natural Language Engineering*, 6(1):63–80, 2000.

[Yamada and Matsumoto, 2003] H. Yamada and Y. Matsumoto. Statistical dependency analysis with support vector machines. In *Proc. IWPT*, pages 195–206, 2003.