Probabilistic models for disambiguation of an HPSG-based chart generator

Hiroko Nakanishi†
†Department of Computer Science
University of Tokyo
Hongo 7-3-1, Bunkyo-ku
Tokyo 113-0033, Japan

Yusuke Miyao†
‡CREST, JST
Honcho 4-1-8, Kawaguchi-shi
Saitama 332-0012, Japan

Jun'ichi Tsujii†‡
‡School of Informatics
University of Manchester
POBox 88, Sackville St
MANCHESTER M60 1QD, UK

n165, yusuke, tsujii@is.s.u-tokyo.ac.jp

Abstract

We describe probabilistic models for a chart generator based on HPSG. Within the research field of parsing with lexicalized grammars such as HPSG, recent developments have achieved efficient estimation of probabilistic models and high-speed parsing guided by probabilistic models. The focus of this paper is to show that two essential techniques – model estimation on packed parse forests and beam search during parsing – are successfully exported to the task of natural language generation. Additionally, we report empirical evaluation of the performance of several disambiguation models and how the performance changes according to the feature set used in the models and the size of training data.

1 Introduction

Surface realization is the final stage of natural language generation which receives a semantic representation and outputs a corresponding sentence where all words are properly inflected and ordered. This paper presents log-linear models to address the ambiguity which arises when HPSG (Head-driven Phrase Structure Grammar (Pollard and Sag, 1994)) is applied to sentence generation. Usually a single semantic representation can be realized as several sentences. For example, consider the following two sentences generated from the same input.

- The *complicated* language in the huge new law has muddied the fight.
- The language in the huge new law *complicated* has muddied the fight.

The latter is not an appropriate realization because *"complicated"* tends to be wrongly interpreted to modify *"law"*. Therefore the generator needs to select a candidate sentence which is more fluent and easier to understand than others.

In principle, we need to enumerate all alternative realizations in order to estimate a log-linear model for generation. It therefore requires high computational cost to estimate a probabilistic model for a wide-coverage grammar because there are considerable ambiguities and the alternative realizations are hard to enumerate explicitly. Moreover, even after the model has been estimated, to explore all possible candidates in runtime is also expensive. The same problems also arise with HPSG parsing, and recent studies (Tsuruoka et al., 2004; Miyao and Tsujii, 2005; Ninomiya et al., 2005) proposed a number of solutions including the methods of estimating log-linear models using packed forests of parse trees and pruning improbable candidates during parsing.

The aim of this paper is to apply these techniques to generation. Since parsing and generation both output the best probable tree under some constraints, we expect that techniques that work effectively in parsing are also beneficial for generation. First, we enabled estimation of log-linear models with less cost by representing a set of generation trees in a packed forest. The forest representation was obtained by adopting chart generation (Kay, 1996; Carroll et al., 1999) where ambiguous candidates are packed into an *equivalence class* and mapping a chart into a forest in the same way as parsing. Second, we reduced the search space in runtime by adopting *iterative beam search* (Tsuruoka and Tsujii, 2004) that efficiently pruned improbable candidates. We evaluated the generator on the Penn Treebank (Marcus et al., 1993), which is highly reliable corpus consisting of real-world texts.

Through a series of experiments, we compared the performance of several disambiguation models following an existing study (Velldal and Oepen, 2005) and examined how the performance changed according to the size of training data, the feature set, and the beam width. Comparing the latter half of the experimental results with those on parsing (Miyao and Tsujii, 2005), we investigated similarities and differences between probabilistic models for parsing and generation. The results indicated that the techniques exported from parsing to generation worked well while the effects were slightly different in detail.

The Nitrogen system (Langkilde and Knight, 1998; Langkilde, 2000) maps semantic relations to a packed forest containing all realizations and selects the best one with a bigram model. Our method extends their approach in that we can utilize syntactic features in the disambiguation model in addition to the bigram.

From the perspective of using a lexicalized grammar developed for parsing and importing parsing techniques, our method is similar to the following approaches. The Fergus system (Bangalore and Rambow, 2000) uses LTAG (Lexicalized Tree Adjoining Grammar (Schabes et al., 1988)) for generating a word lattice containing realizations and selects the best one using a trigram model. White and Baldridge (2003) developed a chart generator for CCG (Combinatory Categorial Grammar (Steedman, 2000)) and proposed several techniques for efficient generation such as best-first search, beam thresholding and chunking the input logical forms (White, 2004). Although some of the techniques look effective, the models to rank candidates are still limited to simple language models. Carroll et al. (1999) developed a chart generator using HPSG. After the generator outputs all the sentences the grammar allows, the ranking mod-

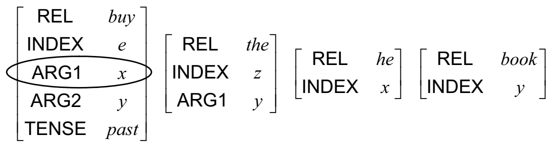

Figure 1: PASs for *"He bought the book."*

ule (Velldal and Oepen, 2005) selects the best one using a log-linear model. Their model is trained using only 864 sentences where all realizations can be explicitly enumerated.

As a grammar is extended to support more linguistic phenomena and to achieve higher coverage, the number of alternative realizations increases and the enumeration requires much higher computational cost. Moreover, using a variety of syntactic features also increases the cost. By representing a set of realizations compactly with a packed forest, we trained the models with rich features on a large corpus using a wide-coverage grammar.

2 Background

This section describes background of this work including the representation of the input to our generator, the algorithm of chart generation, and probabilistic models for HPSG.

2.1 Predicate-argument structures

The grammar we adopted is the Enju grammar, which is an English HPSG grammar extracted from the Penn Treebank by Miyao et al. (2004). In parsing a sentence with the Enju grammar, semantic relations of words is output included in a parse tree. The semantic relations are represented by a set of *predicate-argument structures* (PASs), which in turn becomes the input to our generator. Figure 1 shows an example input to our generator which corresponds to the sentence *"He bought the book."*, which consists of four predicates. REL expresses the base form of the word corresponding to the predicate. INDEX expresses a *semantic variable* to identify each word in the set of relations. ARG1 and ARG2 express relationships between the predicate and its arguments, e.g., the circled part in Figure 1 shows *"he"* is the subject of *"buy"* in this example. The other constraints in the parse tree are omitted in the input for the generator. Since PASs abstract away superficial differences, generation from a set of PASs contains ambiguities in the order of modifiers like the example in Section 1 or the syntactic categories of phrases. For example, the PASs in Figure 1 can generate the NP, *"the book he bought."*

When processing the input PASs, we split a single PAS into a set of relations like (1) representing the first PAS in Figure 1.

$$\mathrm{buy}(e) \wedge \mathrm{past}(e) \wedge \mathrm{ARG1}(e, x) \wedge \mathrm{ARG2}(e, y) \quad (1)$$

This representation is very similar to the notion of HLDS (Hybrid Logic Dependency Semantics) employed by White and Baldridge (2003), which is a related notion to MRS (Minimal Recursion Semantics) employed by Carroll et al. (1999). The most significant difference between our current input representation (not PAS itself) and the other representations is that each word corresponds to exactly one PAS while words like infinitival *"to"* have no semantic relations in HLDS. This means that *"The book was bought by him."* is not generated from the same PASs as Figure 1 because there must be the PASs for *"was"* and *"by"* to generate the sentence.

We currently adopt this constraint for simple implementation, but it is possible to use the input where PASs for words like *"to"* are removed. As proposed and implemented in the previous studies (Carroll et al., 1999; White and Baldridge, 2003), handling such inputs is feasible by modification in chart generation described in the following section. The algorithms proposed in this paper can be integrated with their algorithms although the implementation is left for future research.

2.2 Chart generation

Chart generation is similar to chart parsing, but what an edge covers is the semantic relations associated with it. We developed a CKY-like generator which deals with binarized grammars including the Enju. Figure 2 shows a chart for generating *"He bought the book."* First, lexical edges are assigned to each PAS. Then the following loop is repeated from $n = 2$ to the cardinality of the input.

- Apply binary rules to existing edges to generate new edges holding n PASs.
- Apply unary rules to the new edges generated in the previous process.
- Store the edges generated in the current loop into the chart[1].

In Figure 2, boxes in the chart represent $cells$, which contain edges covering the same PASs, and solid arrows represent rule applications. Each edge is packed into an equivalence class and stored in a cell. Equivalence classes are identified with their signs and the semantic relations they cover. Edges with different strings (e.g., NPs associated with *"a big white dog"* and *"a white big dog"*) can be packed into the same equivalence class if they have the same feature structure.

In parsing, each edge must be combined with its adjacent edges. Since there is no such constraint in generation, the combinations of edges easily explodes. We followed two partial solutions to this problem by Kay (1996).

The one is indexing edges with the semantic variables (e.g., circled y in Figure 2). For example, since the SUBCAT feature of the edge for *"bought the book"* specifies that it requires an NP with an index x, we can find the required edges efficiently by checking the edges indexed with x.

The other is prohibiting proliferation of grammatically correct, but unusable sub-phrases. During generating the sentence "Newspaper reports said that the tall young Polish athlete ran fast", sub-phrases with incomplete modifiers such as "the tall young athlete" or "the young Polish athlete" do not construct the final output, but slow down the generation because they can be combined with the rest of the input to construct grammatically correct phrases or sentences. Carroll et al. (1999) and White (2004) proposed different algorithms to address the same problem. We adopted Kay's simple solution in the current ongoing work, but logical form chunking proposed by White is also applicable to our system.

2.3 Probabilistic models for generation with HPSG

Some existing studies on probabilistic models for HPSG parsing (Malouf and van Noord, 2004; Miyao and Tsujii, 2005) adopted log-linear models (Berger et al., 1996). Since log-linear models allow us to

[1]To introduce an edge with no semantic relations as mentioned in the previous section, we need to combine the edges with edges having no relations.

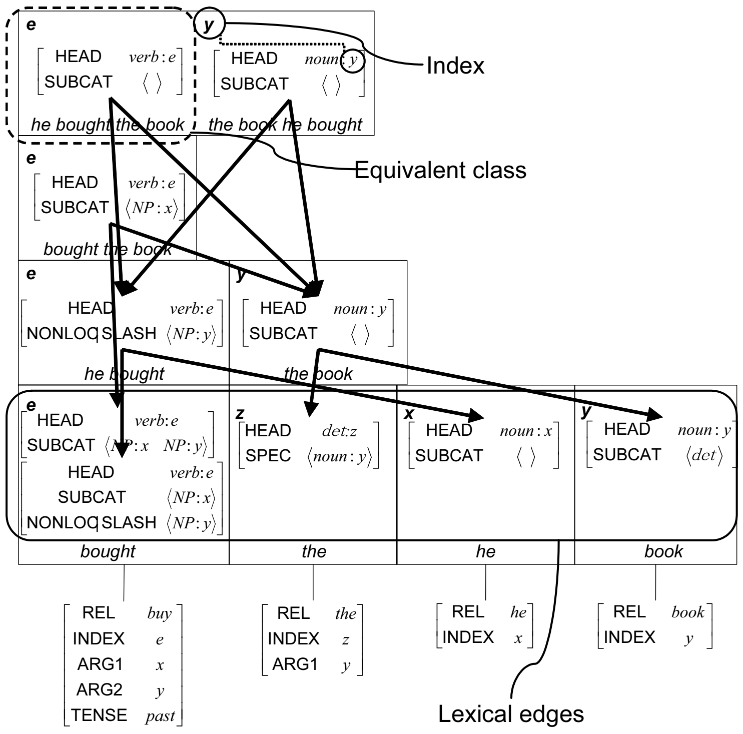

Figure 2: The chart for *"He bought the book."*

use multiple overlapping features without assuming independence among features, the models are suitable for HPSG parsing where feature structures with complicated constraints are involved and dividing such constraints into independent features is difficult. Log-linear models have also been used for HPSG generation by Velldal and Oepen (2005). In their method, the probability of a realization r given a semantic representation s is formulated as

$$p_\lambda(r|s) = \frac{\exp(\sum_i \lambda_i f_i(r))}{\sum_{r' \in Y(s)} \exp(\sum_i \lambda_i f_i(r'))},$$

where $f_i(r)$ is a feature function observed in r, λ_i is the weight of f_i, and $Y(s)$ represents the set of all possible realizations of s. To estimate λ_i, pairs of $\langle r, Y(s) \rangle$ are needed, where r is the most preferred realization for s. Their method first automatically generates a paraphrase treebank, where $\langle r, s, Y(s) \rangle$ are enumerated. Then, a log-linear model is trained with this treebank, i.e., each λ_i is estimated so as to maximize the likelihood of training data. As well as features used in their previous work on statistical parsing (Toutanova and Manning, 2002), an additional feature that represents sentence probabilities of 4-gram model is incorporated. They showed that the combined model outperforms the model without the 4-gram feature.

3 Disambiguation models for chart generation

3.1 Packed representation of a chart

As mentioned in Section 2.3, to estimate log-linear models for HPSG generation, we need all alternative derivation trees $T(s)$ generated from the input s. However, the size of $T(s)$ is exponential to the cardinality of s and they cannot be enumerated explicitly. This problem is especially serious in wide-coverage grammars because such grammars are designed to cover a wide variety of linguistic phenomena, and thus produce many realizations. In this section, we present a method of making the estimation tractable which is similar to a technique developed for HPSG parsing.

When estimating log-linear models, we map $T(s)$ in the chart into a packed representation called a *feature forest*, intuitively an "AND-OR" graph. Miyao and Tsujii (2005) represented a set of HPSG parse

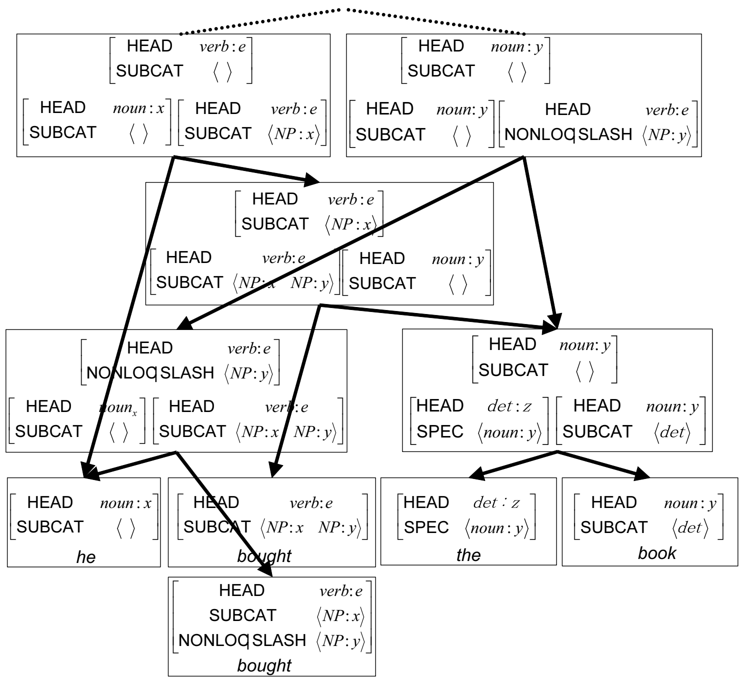

Figure 3: Feature forest for *"He bought the book."*

trees using a feature forest and succeeded in estimating $p(t|w)$ given a sentence w and a parse tree t using dynamic programming without unpacking the chart. If $T(s)$ is represented in a feature forest in generation, $p(t|s)$ can also be estimated in the same way.

Figure 3 shows a feature forest representing the chart in Figure 2. Each node corresponds to either a lexical entry or a tuple of $\langle e_m, e_l, e_r \rangle$ where e_m, e_l and e_r are respectively the mother edge, the left daughter, and the right daughter in a single rule application. Nodes connected by dotted lines represent OR-nodes, i.e., equivalence classes in the same cell. Feature functions are assigned to OR-nodes. By doing so, we can capture important features for disambiguation in HPSG, i.e., combinations of a mother and its daughter(s). Nodes connected by solid arrows represent AND-nodes corresponding to the daughters of the parent node. By using feature forests, we can efficiently pack the node generated more than once in the set of trees. For example, the nodes corresponding to *"the book"* in *"He bought the book."* and *"the book he bought"* are identical and described only once in the forest. The merits of using forest representations in generation instead of lattices or simple enumeration are discussed thoroughly by Langkilde (2000).

3.2 Model variation

We implemented and compared four different disambiguation models as Velldal and Oepen (2005) did. Throughout the models, we assigned a score called *figure-of-merit* (FOM) on each edge and calculated the FOM of a mother edge by dynamic programming. FOM represents the log probability of an edge which is not normalized.

Baseline model We started with a simple baseline model, $p(t|s) = \prod_{r \in s} p(l|r)$, where $r \in s$ is a PAS in the input semantic representation s and l is a lexical entry assigned to r. The FOM of the mother edge $\Delta(e_m)$ is computed simply as $\Delta(e_m) = \Delta(e_l) + \Delta(e_r)$. All the other models use this model as a reference distribution (Miyao and Tsujii, 2005), i.e., λ_i is estimated to maximize the likelihood of the training data $\tilde{p}$, which is calculated with the following equation.

$$\tilde{p}_\lambda(t|s) = \prod_{r \in s} p(l|r) \cdot \frac{\exp(\sum_i \lambda_i f_i(t))}{\sum_{t' \in T(s)} \exp(\sum_i \lambda_i f_i(t'))}$$

Bigram model The second model is a log-linear model with only one feature that corresponds to bigram probabilities for adjacent word-pairs in the sentence. We estimated a bigram language model using a part of the British National Corpus as training data[2]. In the chart each edge is identified with the first and last word in the phrase as well as its feature structure and covered relations. When two edges are combined, $\Delta(e_m)$ is computed as $\Delta(e_m) = \Delta(e_l) + \Delta(e_r) - \lambda p_{bigram}(w_r|w_l)$, where λ is the weight of the bigram feature, w_l is the last word of the left daughter, w_r is the first word of the right daughter, and p_{bigram} represents a log probability of a bigram. Contrary to the method of Velldal and Oepen (2005) where the input is a set of sentences and p_{n-gram} is computed on a whole sentence, we computed p_{bigram} on each phrase as Langkilde (2000) did The language model can be extended to n-gram if each edge holds last $n-1$ words although the number of edges increase.

Syntax model The third model incorporates a variety of syntactic features and lexical features where $\Delta(e_m)$ is computed as $\Delta(e_l) + \Delta(e_r) + \sum_i \lambda_i f_i(e_m, e_l, e_r)$. The feature set consists of combinations of atomic features shown in Table 1. The atomic features and their combinations are imported from the previous work on HPSG parsing (Miyao and Tsujii, 2005). We defined three types of feature combinations to capture the characteristics of binary and unary rule applications and root edges as described below.

$$f_{binary} = \left\langle \begin{array}{l} \mathrm{RULE}, \mathrm{DIST}, \mathrm{COMMA}, \\ \mathrm{SPAN_l}, \mathrm{SYM_l}, \mathrm{WORD_l}, \mathrm{LE_l}, \\ \mathrm{SPAN_r}, \mathrm{SYM_r}, \mathrm{WORD_r}, \mathrm{LE_r} \end{array} \right\rangle$$

$$f_{unary} = \langle \mathrm{RULE}, \mathrm{SYM}, \mathrm{WORD}, \mathrm{LE} \rangle$$

$$f_{root} = \langle \mathrm{SYM}, \mathrm{WORD}, \mathrm{LE} \rangle$$

An example of extracted features is shown in Figure 4 where *"bought the book"* is combined with its subject *"he"*. Since the mother edge is a root edge, two features (f_{root} and f_{binary}) are extracted from this node. In the f_{root} feature, the phrasal category SYM becomes S (sentence), the head word WORD becomes *"bought"*, and its lexical entry LE becomes that of transitive verbs. In the f_{binary} feature, properties of the left and right daughters are instantiated in addition to those of the mother edge.

[2]The model estimation was done using the CMU-Cambridge Statistical Language Modeling toolkit (Clarkson and Rosenfeld, 1997).

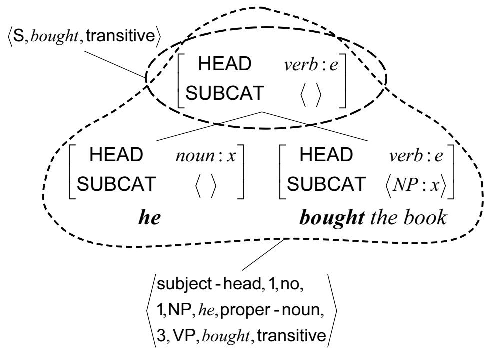

Figure 4: Example of features

Table 1: Atomic features

RULE	the name of the applied schema
DIST	the distance between the head words of the daughters
COMMA	whether a comma exists between daughters and/or inside of daughter phrases
SPAN	the number of words dominated by the phrase
SYM	the symbol of the phrasal category (e.g., NP, VP)
WORD	the surface form of the head word
LE	the lexical entry assigned to the head word

Combined model The fourth and final model is the combination of the syntax model and the bigram model. This model is obtained by simply adding the bigram feature to the syntax model.

4 Iterative beam search

For efficient statistical generation with a wide-coverage grammar, we reduce the search space by pruning edges during generation. We use beam search where edges with low FOMs are pruned during generation. We use two parameters, n and d: in each cell, the generator prunes except for top n edges, and edges whose FOMs are lower than that of the top edge d are also pruned.

Another technique for achieving efficiency is *iterative generation* which is adopted from iterative CKY parsing (Tsuruoka and Tsujii, 2004). When beam width is too narrow, correct edges to constitute a correct sentence may be discarded during gen-

Table 2: Averaged generation time and accuracy by four models

Model		Baseline	Bigram	Syntax	Combined
Coverage (%)		91.15	90.15	90.75	90.56
Time (ms)		3512	4085	3821	4315
BLEU	$\mathrm{length} < 5$ (89 sentences)	0.7776	0.7503	0.8195	0.7359
	$5 \leq \mathrm{length} < 10$ (179 sentences)	0.5544	0.6323	0.7339	0.7305
	$10 \leq \mathrm{length} < 15$ (326 sentences)	0.5809	0.6415	0.7735	0.7384
	$15 \leq \mathrm{length} < 20$ (412 sentences)	0.5863	0.6542	0.7835	0.7533
	Total (1,006 sentences)	0.5959	0.6544	0.7733	0.7420

eration and it causes degradation in coverage, i.e., the ratio the generator successfully outputs a sentence. The appropriate beam width depends on inputs and cannot be predefined. In iterative generation, the process of chart generation is repeated with increasing beam width until a complete sentence is generated or the beam width exceeds the predefined maximum.

5 Experiments

In this section, we present five experiments: comparison among four models described in Section 3.2, syntax models with different features, different corpus sizes, different beam widths, and the distribution of generation time. The bigram model was trained using 100,000 sentences in the BNC. The unigram and syntax model was trained using Section 02-21 of the WSJ portion of the Penn Treebank (39,832 sentences). Section 22 (1,700 sentences) and 23 (2,416 sentences) were used as the development and test data, respectively.

Because the generator is still slow to generate long sentences, sentences with more than 20 words were not used. We converted the treebank into HPSG-style derivation trees by the method of Miyao et al. (2004) and extracted the semantic relations, which are used as the inputs to the generator. The sentences where this conversion failed were also eliminated although such sentences were few (about 0.3% of the eliminated data). The resulting training data consisted of 18,052 sentences and the test data consisted of 1,006 sentences. During training, *uncovered* sentences – where the lexicon does not include the lexical entry to construct correct derivation – were also ignored, while such sentences remained in the test data. The final training data we can utilize consisted of 15,444 sentences. The average sentence length of the test data was 12.4, which happens to be close to that of Velldal and Oepen (2005) though the test data is different.

The accuracy of the generator outputs was evaluated by the BLEU score (Papineni et al., 2001), which is commonly used for the evaluation of machine translation and recently used for the evaluation of generation (Langkilde-Geary, 2002; Velldal and Oepen, 2005). BLEU is the weighted average of n-gram precision against the reference sentence. We used the sentences in the Penn Treebank as the reference sentences. The beam width was increased from $(n, d) = (8, 4.0)$ to $(20, 10.0)$ in two steps. The parameters were empirically determined using the development set. All the experiments were conducted on AMD Opteron servers with a 2.0-GHz CPU and 12-GB memory.

Table 2 shows the average generation time and the accuracy of the models presented in Section 3. The generation time includes time for the input for which the generator could not output a sentence, while the accuracy was calculated only in the case of successful generation. All models succeeded in generation for over 90% of the test data.

Contrary to the result of the Velldal and Oepen (2005), the syntax model outperformed the combined model. We observed the same result when we varied the parameters for beam thresholding. This is possibly just because the language model was not trained enough as that of the previous research (Velldal and Oepen, 2005) where the model was 4-gram and trained with the entire BNC[3].

[3]We could not use the entire corpus for training because of a problem in implementation. This problem will be fixed in the

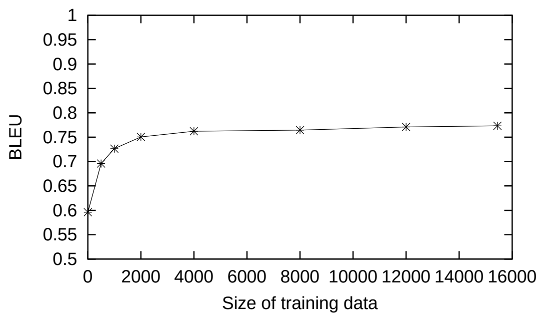

Figure 5: Size of training data vs. performance

Table 3: Feature set vs. performance

Feature	BLEU	diff.
All	0.7734	
-COMMA	**0.7492**	**-0.0242**
-DIST	0.7702	-0.0032
-LE	**0.7423**	**-0.0311**
-RULE	0.7709	-0.0025
-SPAN	0.7640	-0.0094
-SYM	**0.7400**	**-0.0334**
-WORD	0.7610	-0.0124
None	0.5959	-0.1775

Although the accuracy shown in Table 2 was lower than that of Velldal and Oepen, there is little point in direct comparison between the accuracy of the two systems because the settings are considerably different in terms of the grammar, the input representation, and the training and test set. The algorithm we proposed does not depend on our specific setting and can be integrated and evaluated within their setting. We used larger training data (15,444 sentences) and test data (1,006 sentences), compared to their treebank of 864 sentences where the log-linear models were evaluated by cross validation. This is the advantage of adopting feature forests to efficiently estimate the log-linear models.

Figure 5 shows the relationship between the size of training data and the accuracy. All the following experiments were conducted on the syntax model. The accuracy seems to saturate around 4000 sentences, which indicates that a small training set is enough to train the current syntax model and that we could use an additional feature set to improve the accuracy. Similar results are reported in parsing (Miyao and Tsujii, 2005) while the accuracy saturated around 16,000 sentences. When we use more complicated features or train the model with longer sentences, possibly the size of necessary training data will increase.

future development.

Table 4: n vs. performance

n	Coverage (%)	Time (ms)	BLEU
4	66.10	768	0.7685
8	82.91	3359	0.7654
12	87.89	7191	0.7735
16	89.46	11051	0.7738
20	90.56	15530	0.7723

Table 5: d vs. performance

d	Coverage (%)	Time (ms)	BLEU
4.0	78.23	2450	0.7765
6.0	89.56	9083	0.7693
8.0	91.15	19320	0.7697
10.0	89.86	35897	0.7689

Table 3 shows the performance of syntax models with different feature sets. Each row represents a model where one of the atomic features in Table 1 was removed. The "None" row is the baseline model. The rightmost column represents the difference of the accuracy from the model trained with all features. SYM, LE, and COMMA features had a significant influence on the performance. These results are different from those in parsing reported by Miyao and Tsujii (2005) where COMMA and SPAN especially contributed to the accuracy. This observation implies that there is still room for improvement by tuning the combination of features for generation.

We compared the performance of the generator with different beam widths to investigate the effect of iterative beam search. Table 4 shows the results when we varied n, which is the number of edges, while thresholding by FOM differences is disabled, and Table 5 shows the results when we varied only d, which is the FOM difference.

Intuitively, beam search may decrease the accuracy because it cannot explore all possible candi-

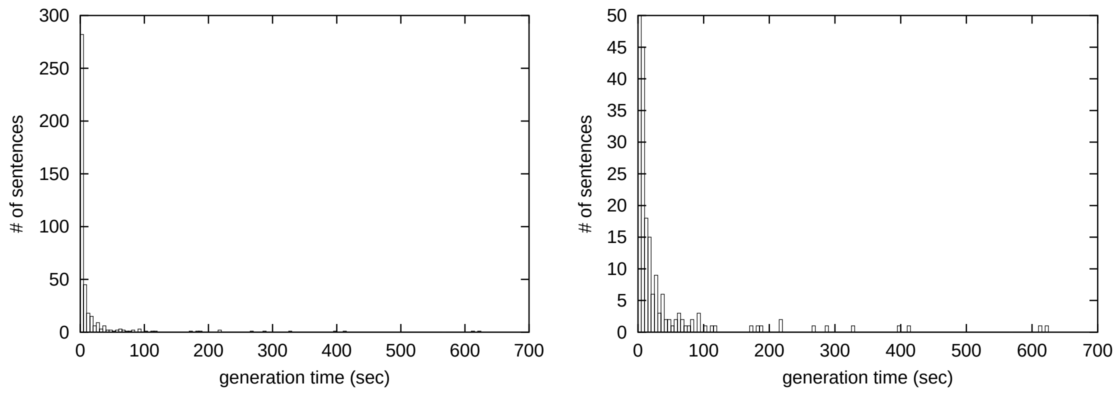

Figure 6: Distribution of generation time

dates during generation. Iterative beam search is more likely to decrease the accuracy than ordinary beam search. However, the results show that the accuracy did not drastically decrease at small widths. Moreover, the accuracy of iterative beam search was almost the same as that of $n = 20$. On the other hand, generation time significantly increased as n or d increased, indicating that iterative beam search efficiently discarded unnecessary edges without loosing the accuracy. Although the coverage increases as the beam width increases, the coverage at $n = 20$ or $d = 10.0$ is lower than that of iterative beam search (Table 2)[4].

Finally, we examined the distribution of generation time without the limitation of sentence length in order to investigate the strategy to improve the efficiency of the generator. Figure 6 is a histogram of generation time for 500 sentences randomly selected from the development set, where 418 sentences were successfully generated and the average BLEU score was 0.705. The average sentence length was 22.1 and the maximum length was 60, and the average generation time was 27.9 sec, which was much longer than that for short sentences. It shows that a few sentences require extremely long time for generation although about 70% of the sentences were generated within 5 sec. Hence, the average time possibly decreases if we investigate what kind of sentences require especially long time and improve the algorithm to remove such time-consuming fractions. The investigation is left for future research.

The closest empirical evaluations on the same task is that of Langkilde-Geary (2002) which reported the performance of the HALogen system while the approach is rather different. Hand-written mapping rules are used to make a forest containing all candidates and the best candidate is selected using the bigram model. The performance of the generator was evaluated on Section 23 of the Penn Treebank in terms of the number of ambiguities, generation time, coverage, and accuracy. Several types of input specifications were examined in order to measure how specific the input should be for generating valid sentences. One of the specifications named "permute, no dir" is similar to our input in that the order of modifiers is not determined at all. The generator produced outputs for 82.7% of the inputs with average generation time 30.0 sec and BLEU score 0.757. The results of our last experiment are comparable to these results though the used section is different.

6 Conclusion

We presented a chart generator using HPSG and developed log-linear models which we believe was essential to develop a sentence generator. Several techniques developed for parsing also worked in generation. The introduced techniques were an estimation method for log-linear models using a packed forest representation of HPSG trees and iterative beam search. The system was evaluated through application to real-world texts. The experimental results

[4]This is because the generator fails when the number of edges exceeds 10,000. Since the number of edges significantly increases when n or d is large, generation fails even if the correct edges are in the chart.

showed that the generator was able to output a sentence for over 90% of the test data when the data was limited to short sentences. The accuracy was significantly improved by incorporating syntactic features into the log-linear model. As future work we intend to tune the feature set for generation. We also plan to further increase the efficiency of the generator so as to generate longer sentences.

References

S. Bangalore and O. Rambow. 2000. Exploiting a probabilistic hierarchical model for generation. In *Proceedings of the COLING'00*.

A. L. Berger, S. A. Della Pietra, and V. J. Della Pietra. 1996. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71.

J. Carroll, A. Copestake, D. Flickinger, and V. Poznanski. 1999. An efficient chart generator for (semi-)lexicalist grammars. In *Proceedings of the EWNLG'99*.

P. Clarkson and R. Rosenfeld. 1997. Statistical language modeling using the CMU-Cambridge toolkit. In *Proceedings of ESCA Eurospeech*.

M. Kay. 1996. Chart generation. In *Proceedings of the ACL'96*, pages 200–204.

I. Langkilde and K. Knight. 1998. Generation that exploits corpus-based statistical knowledge. In *Proceedings of the COLING-ACL'98*, pages 704–710.

I. Langkilde-Geary. 2002. An empirical verification of coverage and correctness for a general-purpose sentence generator. In *Proceedings of the INLG'02*.

I. Langkilde. 2000. Forest-based statistical sentence generation. In *Proceedings of the NAACL'00*.

R. Malouf and G. van Noord. 2004. Wide coverage parsing with stochastic attribute value grammars. In *Proceedings of the IJCNLP-04 Workshop on Beyond Shallow Analyses*.

M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*.

Y. Miyao and J. Tsujii. 2005. Probabilistic disambiguation models for wide-coverage HPSG parsing. In *Proceedings of the ACL'05*.

Y. Miyao, T. Ninomiya, and J. Tsujii. 2004. Corpus-oriented grammar development for acquiring a Head-Driven Phrase Structure Grammar from the Penn Treebank. In *Proceedings of the IJCNLP-04*.

T. Ninomiya, Y. Tsuruoka, Y. Miyao, and J. Tsujii. 2005. Efficacy of beam thresholding, unification filtering and hybrid parsing in probabilistic HPSG parsing. In *Proceedings of the IWPT'05*.

K. Papineni, S. Roukos, T. Ward, and W. Zhu. 2001. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the ACL'01*.

C. Pollard and I. A. Sag. 1994. *Head-driven Phrase Structure Grammar*. University of Chicago Press.

Y. Schabes, A. Abeille, and A. K. Joshi. 1988. Parsing strategies with 'lexicalized' grammars: Application to tree adjoining grammar. In *Proceedings of the COLING 1988*, pages 578–583.

M. Steedman. 2000. *The Syntactic Process*. MIT Press.

K. Toutanova and C. D. Manning. 2002. Feature selection for a rich HPSG grammar using decision trees. In *Proceedings of the CoNLL'02*.

Y. Tsuruoka and J. Tsujii. 2004. Iterative CKY parsing for Probabilistic Context-Free Grammars. In *Proceedings of the IJCNLP'04*.

Y. Tsuruoka, Y. Miyao, and J. Tsujii. 2004. Towards efficient probabilistic HPSG parsing: integrating semantic and syntactic preference to guide the parsing. In *Proceedings of the IJCNLP-04 Workshop on Beyond Shallow Analyses*.

E. Velldal and S. Oepen. 2005. Maximum entropy models for realization ranking. In *Proceedings of the MT-Summit'05*.

M. White and J. Baldridge. 2003. Adapting chart realization to CCG. In *Proceedings of the EWNNLG'03*.

M. White. 2004. Reining in CCG chart realization. In *Proceedings of the INLG'04*.