Using Dependency Relations as Constraints in HPSG Parsing

Eric Nichols and Yuji Matsumoto
Graduate School of Information Science
Nara Institute of Science and Technology
{eric-n,matsu}@is.naist.jp

Abstract

In this paper, we present a novel chart parsing algorithm, called the “filtering algorithm”, which was designed for use in a Head-driven Phrase Structure Grammar (HPSG) analysis system for Japanese. The filtering algorithm uses information from dependency trees to constrain the generation of candidate phrase-structure trees for HPSG analysis which leads to a reduction in parse ambiguity and an increase in parser effeciency. We present a generalization of this algorithm that is independent of the order that new items are added to the chart and to the type of grammar that is used to create new chart entries. We provide evaluation of this parser’s accuracy, coverage, and ambiguity reduction capability by parsing sentences from the Hinoki Treebank, a treebank of hand-verified Japanese HPSG analyses, and comparing its results with those of the PET System used in its construction.

1 Introduction

As advances in both the state-of-the-art in computing power and the field of linguistics have permitted, deep processing techniques have continued to gain attention in the field of natural language processing. Syntactic theories such as Head-driven Phrase Structure Grammar (HPSG) [10] and Lexical-Functional Grammar (LFG) provide a wealth of linguist information that is useful in high-level tasks such as machine translation (LOGON: `http://www.emmtee.net/`) and ontology acquisition [3].

However, deep processing has its own problematic areas. One of particular concern is dealing with the large number of analyses that can be generated by such a system. Often times, hundreds or even thousands of parses can be generated for a single sentence. Devising methods of dealing with this staggering amount of ambiguity has been an active are of research. A good overview of these methods can be found in [12] where the authors classified methods into *reranking* and *dynamic programming* subgroups. Reranking techniques were characterized with allowing the parser to generate all possible parse candidates and using a probabilistic model to identify the desired analysis from among predetermined number of trees. Dynamic programming techniques, on the other hand, perform similar probabilistic evaluation over all of parses when they are still stored in a chart or parse forest. There is another class of techniques not mentioned in the above research: pruning results or guiding a parser using a Context-Free Grammar (CFG) [13, 6].

The aforementioned probabilistic models have the disadvantage of being limited to a small number of features or being too computationally demanding to be practical. On the other hand, CFG-based pruning or parse guidance is impractical because it requires a grammar to be hand-constructed or induced, and areas of disagreement between the CFG and HPSG grammar result in loss of coverage.

In response to this, we propose that information from dependency trees be used to constrain generation of unlikely trees. By using a robust, statistical dependency analyzer to filter HPSG parses, we can reduce parse ambiguity while reducing the computational cost of parsing and maintaining a high level of coverage.

We proposed a Japanese HPSG parser in [9, 1] that employs the output from a statistical dependency analyzer together with a simple, hand-crafted CFG to give a uniform analysis for scrambled sentences containing compound verbs. The chart parsing algorithm that was at the core of this system also showed potential for reducing parse output ambiguity. In this paper, we further this chart parsing algorithm, hereafter referred to as the “filtering algorithm”, to be applicable to any kind of binary chart parsing algorithm and provide concrete evaluation of its effectiveness at decreasing parse output ambiguity and improving parser efficiency by comparing its performance to that of the Cheap parser [4] on the Hinoki Treebank [2].

2 Filtering Algorithm

The filtering algorithm is an extension of a chart parsing algorithm such as that presented in [5]. However, it is generalized to be independent of the grammar used to produce analyses and the algorithm that determines the order that new items are added to the chart. In essence, the filtering algorithm is a set of constraints that wraps around the portion of the chart parsing algorithm in charge of making additions to the chart, allowing access only when the chart items under consideration meet criteria determined by the dependency tree corresponding to the input sentence.

Thus, the object is to only allow new entries to be added to the chart when they reflect relations that are present in the dependency analysis of the sentence being parsed. There are two kinds of relations present in dependency trees: parent-child relations and relations internal to a dependency node (also known as a “chunk”). I will now describe how an algorithm can be constructed that will only add a chart entry when one of the aforementioned relations is true.

Let us assume the following terminology:

- *left_edge*: the leftmost of two edges being considered together for combination inn order to produce a new chart entry
- *right_edge*: the rightmost of these two edges
- **chunk**(*edge*): a function which takes an edge as input and returns an identifier of the dependency node that *edge* corresponds to in the dependency structure
- **parent**(*edge*): a function which takes an edge as input, looks up its dependency node using **chunk()**, and returns an identifier corresponding to the parent of edge’s node in the dependency, and NULL if edge is in the root of the dependency tree
- **coverage**(*edge*): a function which calculates the range of input sentence covered by *edge*
- **covers_children**(*edge*): a function which returns *true* if all children nodes of **chunk**(*edge*) are subsumed by **coverage**(*edge*), returns *false* otherwise

A Cooke Kasami Young (CKY) chart parsing algorithm may have a representation similar to the function **cky_scheduler()** given in Figure 2. This can be augmented with the filtering algorithm simply by replacing the chart entry creation function, **combine_edges()**, with **filtering_edge_combiner()**. **combine_edges()** is a function pointer which changes depending on the type of the chart to point to a function that performs whatever actions are necessary to attempt to add a new entry to a chart. For example, if the chart used a CFG grammar, then **combine_edges()** would most likely point to a function like **apply_cfg_rule()** which would check for a CFG rule of the format *[new_edge* → *left_edge right_edge]* and would add *new_edge* to the chart upon success.

3 Resources

3.1 Parsing Resources

3.1.1 Chart Parser

We used the PET System [4] as a platform to implement and test the filtering algorithm. The PET System is a parser for unification grammars that was designed to facilitate research in efficient parsing techniques and allow for the practical application of such grammars in research. At its core, it uses an adgenda-driven bidirectional chart parser called Cheap. The pseudocode for Cheap’s parsing algorithm as augmented by the filtering algorithm is given in Figure 1. Cheap does not use CFG rules to filter input the the HPSG unifier. As a result of this, **combine_edges()** directly references a function which attempts to unify passive and active edges with an HPSG grammar and pushes any successful results directly into the chart.

```
cheap_algorithm:

for all left_edge,right_edge where
      (adjacent(left_edge,right_edge))
{
  filtering_edge_combiner(left_edge,right_edge)
}

edge_combiner:

unify_with_hpsg_grammar(left_edge,right_edge)
```

Figure 1: Cheap parser with the filtering algorithm

3.1.2 Dependency Analyzer

Relative CaboCha (rCaboCha) [8, 7] is a Japanese dependency analyzer that uses machine learning techniques to learn dependencies from training data. Relative CaboCha can also produce multiple dependency trees as output, ranking them in order of highest probability. We used this feature to evaluate differences in the filtering algorithm’s using “n-best” parsers for an input sentence. In our evaluation, we compared results using 1, 3, and 5 dependency trees for each input sentence.

3.2 Test Corpus

3.2.1 The Hinoki Treebank

The Hinoki Treebank [2] is a collection of Japanese HPSG analyses. It was constructed by parsing definition sentences from an electronic dictionary using the JACY Japanese grammar [11]. It uses a closed vocabulary of 28,000 words that were ranked “most familiar” by knowledgable native speaker judgement, and its parse results are hand-verified, creating a gold standard of correct analyses that can be used as a metric of comparison.

4 Evaluation and Results

Evaluation is done in three separate experiments:

1. Parse ambiguity reduction: compare the number of HPSG analyses produced per sentence
2. Parse efficiency: compare tasks executed, parser runtime, and memory consumption

Parse Ambiguity (parses per sentence)

Definitions	Sentences	Baseline	1-Best	3-Best	5-Best
Noun	3477	171.61	11.71 (6.83%)	31.88 (18.58%)	52.55 (30.62%)
Verb	3540	19.58	3.16 (16.12%)	5.37 (27.41%)	7.40 (37.77%)
Verbal Noun	2685	127.06	7.63 (6.01%)	14.03 (11.04%)	28.94 (22.78%)
Total	9702	103.81	7.60 (7.32%)	18.96 (18.26%)	29.54 (28.46%)

Table 1: Comparison of average parse ambiguity

Tasks Executed (average tasks per sentence)

Definitions	Sentences	Baseline	1-Best	3-Best	5-Best
Noun	3477	32590	**2471 (7.58%)**	4587 (14.07%)	6924 (21.25%)
Verb	3540	2704	**777 (28.74%)**	1023 (37.83%)	1228 (45.41%)
Verbal Noun	2685	16288	**1461 (8.97%)**	2542 (15.61%)	3310 (20.32%)
Total	9702	51582	**4709 (9.13%)**	8152 (15.80%)	11462 (22.22%)

Parser Runtime (average seconds per sentence)

Definitions	Sentences	Baseline	1-Best	3-Best	5-Best
Noun	3477	0.52	**0.36 (70.86%)**	1.69 (328.12%)	4.36 (846.48%)
Verb	3540	0.07	**0.08 (107.43%)**	0.20 (273.14%)	0.40 (549.88%)
Verbal Noun	2685	0.26	**0.20 (76.53%)**	0.83 (313.41%)	1.83 (695.42%)
Total	9702	0.28	**0.21 (75.23%)**	0.91 (320.00%)	2.21 (780.31%)

Memory Usage (average kilobytes per sentence)

Definitions	Sentences	Baseline	1-Best	3-Best	5-Best
Noun	3477	10563	**2328 (22.04%)**	3099 (29.34%)	3949 (37.39%)
Verb	3540	1810	**1155 (63.78%)**	1241 (68.58%)	1317 (72.79%)
Verbal Noun	2685	5627	**1711 (30.41%)**	2111 (37.52%)	2415 (42.91%)
Total	9702	6003	**1729 (28.81%)**	2148 (35.77%)	2564 (42.71%)

Table 2: Comparison of parser efficiency

Overall Coverage of Definition Sentences

Parser Settings	Sentences Covered	Percent Covered	Coverage Gain
Baseline	6171 / 6344	97.27%	–
1-Best Filtering	5103 / 6344	80.43%	+1.59%
3-Best Filtering	5493 / 6344	86.58%	**+6.15%**
5-Best Filtering	**5561 / 6344**	**87.66%**	+1.08%

Table 3: Comparison of definition sentence coverage

```
chunk_internal_rel(left_edge,right_edge):

if (adjacent(left_edge, right_edge) &&
    (chunk(left_edge) == chunk(right_edge)))
  return true
else
  return false

parent_child_rel(left_edge,right_edge):

if (parent(left_edge) == chunk(right_edge))
  return true
else
  return false
```

```
filtering_edge_combiner(left_edge,right_edge):

if (chunk_internal_rel(left_edge,right_edge) ||
    (parent_child_rel(left_edge,right_edge) &&
     covers_children(left_edge)))

cky_scheduler:

for all <left_edge,right_edge> where
       (adjacent(left_edge,right_edge))
{
  combine_edges(left_edge,right_edge)
}
```

Figure 2: Pseudocode for the filtering algorithm

3. Coverage: compare parse results to gold standard

Testing was done using treebank profiles of noun, verb, and verbal noun definition sentences from the Hinoki Treebank. A total of 9,702 test sentences were used in evaluating performance. Of those sentences, 6,344 have human-verified results and were used in evaluating coverage. There sentences were parsed once with an unmodified Cheap to create a baseline for comparison. The treebank was then parsed again with Cheap using the filtering algorithm with 1-best, 3-best, and 5-best dependency tree input. All testing was done on a computer with a 1.8 Ghz Pentium M processor and 1.5 gigabytes of RAM. Cheap was run with a limit of 1.5 gigabytes of memory and 100,000 active edges for each input sentence.

5 Discussion

There is a more in-depth discussion of these experiments and their results in [9].

5.1 Parse Ambiguity Reduction

Results are given in Table 1. The filtering algorithm showed itself to be quite effective at reducing parse ambiguity. It reduced the baseline average of 103.81 analyses per sentence to a low of 7.55 analyses using 1-best filtering.

5.2 Parse Effi ciency

Results are given in Table 2. The experiments for Tasks Executed and Memory Usage provide fair evidence of the increase in parsing efficiency that the filtering algorithm can provide. On average, memory consumption is reduced from an average of 6,004 kilobytes per sentence to between 28.38 and 42.71 percent. The average number of tasks executed per sentence is likewise reduced from 51,582 tasks to from 9.13 to 22.22 percent of that figure.

Runtime efficiency is the only area where favorable results are not had. 1-best filtering shows a small improvement in speed over the baseline, however, 3-best rCaboCha takes over three times longer than the baseline per sentence, and the runtime of the five-best approaches a magnitude of eight higher. This is most likely due to the current implementation's reliance on the CaboCha library to read multiple dependency trees from file. In doing so, it must create a parser and reconstruct all of the internal data items associated with it. Switching to an I/O format such as XML that is supported by both rCaboCha and Cheap could lead to a significant speedup.

5.3 Coverage

Results are given in Table 3, As expected, the baseline (i.e. the unmodified Cheap parser) has the highest coverage. In theory, the baseline should achieve a perfect coverage of the treebank because the treebank was produced using Cheap with the same settings as used in this experiment. The small loss in coverage can be attributed to differences in lexicons. The authors did not have access to the same version of the tokenizer's lexicon at the time of testing. This resulted in different segmentation of some compound nouns and verbal inflections, leading to misparses for a small number of sentences. Of the dependency parsers, the best performance was 5-best filtering with almost 88% coverage. However, even using only the best parse, rCaboCha still achieves an overall coverage of over 80%. Is is also significant to note that the biggest increase in coverage occurs in moving from 1-best to 3-best filtering; moving to 5-best filtering offers comparably little gain.

6 Conclusion

The filtering parser appears to be effective at reducing parse ambiguity and increasing efficiency while maintaining a high level of coverage. It is interesting to observe that the benefits for using filtering based on n-best results from dependency analysis seems to peak at three results. The overall level of coverage as well as savings in CPU usage and memory consumption for 3-best should make this approach attractive for applications that need to reduce the amount of ambiguity in deep parsing results before they are usable. Future research will focus on improving the current implementation to address the speed issues encountered when using more than one dependency tree and investigating the application of this technique in languages beyond Japanese.

References

[1] A. Azuma, E. Nichols, Y. Morimoto, and Y. Matsumoto. Integration of statistical dependency parsing and constraint based grammar for japanese sentence analysis. In *2004-NL-159*, volume 1, pages 131–138, Tokyo, 2004. (in Japanese).

[2] F. Bond, S. Fujita, C. Hashimoto, K. Kasahara, S. Nariyama, E. Nichols, A. Ohtani, T. Tanaka, and S. Amano. The Hinoki treebank: A treebank for text understanding. In *Proceedings of the First International Joint Conference on Natural Language Processing (IJCNLP-04)*, pages 554–559, Hainan Island, 2004.

[3] F. Bond, E. Nichols, S. Fujita, and T. Tanaka. Acquiring an ontology for a fundamental vocabulary. pages 1319–1325, Geneva, 2004.

[4] U. Callmeier. Preprocessing and encoding techniques in PET. In S. Oepen, D. Flickinger, J. Tsujii, and H. Uszkoreit, editors, *Collabarative Language Engineering*, chapter 6, pages 127–143. CSLI Publications, Stanford, 2002.

[5] M. Kay. Algorithm schemata and data structures in syntactic processing. Technical report, Xerox Corporation, 1980.

[6] B. Kiefer, H.-U. Krieger, and M. Siegel. An hpsg-to-cfg approximation of japanese. 2000.

[7] T. Kudo and Y. Matsumoto. Japanese dependency analyisis using cascaded chunking. Taipei, 2002.

[8] T. Kudo and Y. Matsumoto. Japanese dependency parsing using relative preference of dependency. 2004. (in Japanese).

[9] E. Nichols. The role of dependency trees in hpsg parse fi ltering. Master's thesis, Nara Institute of Science and Technology, 2005. (under consideration).

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

[11] M. Siegel and E. M. Bender. Effi cient deep processing of Japanese. In *Procedings of the 3rd Workshop on Asian Language Resources and International Standardization at the 19th International Conference on Computational Linguistics*, Taipei, 2002.

[12] B. Taskar, D. Klein, M. Collins, D. Koller, and C. Manning. Max-margin parsing. Barcelona, 2004.

[13] Y. Tsuroka, Y. Miyao, and J. Tsujii. Towards effecient probabilistic hpsg parsing. Sanya, 2004.