Approximate Parsing with "Hedge Grammars"

Mahsa Yarmohammadi
Center for Spoken Language Understanding
Oregon Health & Science University
yarmoham@ohsu.edu

1 Introduction

Exact inference with full hierarchical syntactic structures is very costly and in fact, some NLP applications such as NP-chunking or Named Entity Recognition, do not need a full syntactic analysis. These are instances of *sequence tagging* or partial parsing that do not annotate full hierarchical structures and as a result they are significantly faster than the approaches that do. These models are often used as an initial pass in parsing pipeline systems (Glaysher and Moldovan, 2006; Djordjevic et al., 2007; Roark et al., 2012). Another variant is supertagging (Bangalore and Joshi, 1999) which assigns a rich tag – an elementary tree of a Lexicalized Tree-Adjoining Grammar – to each word of a sentence and can be used to prune the (context-sensitive) parser's search space.

Although sequence tagging models have been widely used in NLP and speech applications, they are finite-state equivalent approaches and generally do not preserve the internal structure of the constituents. In some applications, we need to model syntactic dependencies more explicitly than finite-state models typically allow, but it is overkill to apply a full context-free model. For example when a fast partial analysis of the input is required, such as in incremental parsing or translating a sentence as we receive the first words of the sentence. In this paper, we introduce a parsing method which is fast due to being a partial parsing, but unlike the previous methods, it finds the internal hierarchical structure of phrases.

We introduce *hedge parsing* as an approach of discovering every constituent of length up to some span of L. We name structures of this type *hedges*. In other words, a hedge is the lowest constituent in the parse tree that covers at most L words. Hedges are sequentially connected to the immediate child of the ROOT of the tree (TOP node). Figure 1 shows an examples of a full parse tree and its hedge transformed parse tree for $L = 7$. Hedge parses can also be explicitly used as features in a probability model such as a segmentation model that favors limited length segments.

In this study, we propose several methods to parse hedge constituents and we examine their accuracy/efficiency tradeoffs. First, we parse the entire sentence as the input, similar to what is done in common parsing methods. Second, instead of parsing the entire sentence, we chunk it into segments that are (most probably) complete hedges, parse the segments, and recombine the results. In both conditions, we use various grammars induced from full or hedge transformed parse trees. We will see that in general, the first set of approaches are more accurate and slower than the second set.

2 Hedge Parsing

The goal of hedge parsing is to annotate the hedge constituents of an input sentence, for which we investigate several approaches in this paper. The brute-force approach is to parse the sentence using a full context-free grammar and then hedge transform the result. This provides a baseline against which we compare the other methods; it has a high accuracy due to rich context information, with a low efficiency. We aim to dramatically improve efficiency upon this baseline while losing a little accuracy as possible.

Another approach is to use a full context-free grammar with a hedge-constrained parser. Since we limit the span of non-terminal labels, we can also greatly reduce the CYK processing time, by eliminating cells which would span more than L words except cells along the periphery of the chart. However, this approach does not work well for hedge parsing due to unmatched grammar with hedge constraints. Accuracy is quite low although the parsing is much faster than the brute-force approach.

To better match the grammar with hedge constraints, we learn a grammar from a hedge transformed treebank and we call it a *hedge grammar*. A hedge grammar is not only smaller and results in a faster parsing than a full context-free grammar, but also it results in much better accuracy than the previous method due to being matched with hedge constraints.

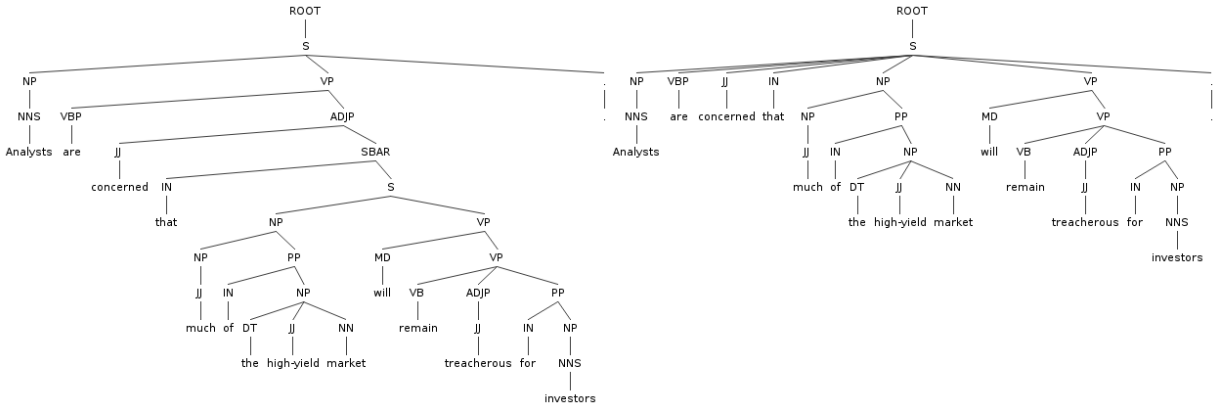

Figure 1: a) Full parse tree, b) Hedge parse tree $L = 7$. Nodes VP, ADJP, SBAR, and the lower S in (a) , which span more than 7 words, are removed and their children are promoted by connecting to the top node S.

A unique property of hedge constituents compared to constituents in the original parse trees is that they are sequentially connected to the TOP node. This property enables us to chunk the sentence into segments which correspond to complete hedges, and parse the segments independently (and simultaneously) instead of parsing the entire sentence. In section 3, we explain our hedge segmentation task. Again, to better match the grammar with segmented inputs, we decompose each tree in the hedge transformed treebank into its hedge segments and learn a grammar from the new corpus which we call it a *hedge-segmented grammar*.

3 Hedge Segmentation

We introduce a segmentation model which gets the input sentence and chunks it into appropriate segments for hedge parsing. In fact, our segmentation model is a binary classifier which decides if a word can begin a new hedge. Given a set of labeled pairs (S, H) where S is a sentence of n words $w_1..w_n$ and H is its hedge parse tree, then word $w_b \in B$ if there is a hedge constituent spanning $w_b..w_e$ for some $e \geq b$ and $w_b \in \bar{B}$ otherwise. We train the binary classifier using a discriminative log-linear model and train the model parameters with the average perceptron algorithm (Collins, 2002). We extract n-gram and orthographical features from word and POS tags sequences.

4 Experiments and Results

We ran all experiments on the WSJ treebank (Marcus et al., 1999) using section 2-21 for training, section 24 for development, and section 23 for testing. We parsed with the Berkeley SM6 latent-variable grammars (Petrov and Klein, 2007) which are learnt by the Berkeley grammar trainer with default settings.

Our results show that parsing the entire input using various grammars is slower but more accurate than parsing the segmented input. In other words, when we parse the segmented input as opposed to the entire input, we trade some accuracy for efficiency. The brute-force approach parses only less than 3 words per second (wps) with a very high accuracy of over 90%. A full context-free grammar with a hedge-constrained parser improves efficiency but decreases accuracy drastically, however, a hedge grammar with this parser improves efficiency upon the brute-force case (26 wps) while keeping the accuracy close to that of the brute-force approach (86.8%).

Segmenting-then-parsing approach is potentially highly accurate and efficient in finding hedge constituents; assuming a perfect input segmentation, we can parse even more accurately than the brute-force approach while we parse around 200 words per second. In practice, we achieve around 85% F1-score in segmentation boundary prediction using our segmentation model. Segmenting the input using this model, parsing the segments using a segmented-hedge grammar, and recombining the results yields to 83% accuracy and over 200 wps efficiency.

References

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

Michael Collins. 2002. Discriminative training methods for hidden Markov models: theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing*, EMNLP '02, pages 1–8, Stroudsburg, PA, USA.

Bojan Djordjevic, James R. Curran, and Stephen Clark. 2007. Improving the efficiency of a wide-coverage CCG parser. In *Proceedings of the 10th International Conference on Parsing Technologies*, IWPT '07, pages 39–47, Stroudsburg, PA, USA.

Elliot Glaysher and Dan I. Moldovan. 2006. Speeding up full syntactic parsing by leveraging partial parsing decisions. In *ACL 2006, 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, Sydney, Australia, 17-21 July 2006*.

Mitchell P. Marcus, Beatrice Santorini, Mary Ann Marcinkiewicz, and Ann Taylor. 1999. Treebank-3. Linguistic Data Consortium, Philadelphia.

Slav Petrov and Dan Klein. 2007. Learning and inference for hierarchically split PCFGs. In *Proceedings of the 22nd national conference on Artificial intelligence - Volume 2*, AAAI'07, pages 1663–1666.

Brian Roark, Kristy Hollingshead, and Nathan Bodenstab. 2012. Finite-state chart constraints for reduced complexity context-free parsing pipelines. *Computational Linguistics*, 38(4):719–753.