Transforming trees into hedges and parsing with "hedgebank" grammars

Mahsa Yarmohammadi[†], **Aaron Dunlop**[†] **and Brian Roark**[◦]
[†]Oregon Health & Science University, Portland, Oregon [◦]Google, Inc., New York
yarmoham@ohsu.edu, {aaron.dunlop,roarkbr}@gmail.com

Abstract

Finite-state chunking and tagging methods are very fast for annotating non-hierarchical syntactic information, and are often applied in applications that do not require full syntactic analyses. Scenarios such as incremental machine translation may benefit from some degree of hierarchical syntactic analysis without requiring fully connected parses. We introduce *hedge parsing* as an approach to recovering constituents of length up to some maximum span L. This approach improves efficiency by bounding constituent size, and allows for efficient segmentation strategies prior to parsing. Unlike shallow parsing methods, hedge parsing yields internal hierarchical structure of phrases within its span bound. We present the approach and some initial experiments on different inference strategies.

1 Introduction

Parsing full hierarchical syntactic structures is costly, and some NLP applications that could benefit from parses instead substitute shallow proxies such as NP chunks. Models to derive such non-hierarchical annotations are finite-state, so inference is very fast. Still, these partial annotations omit all but the most basic syntactic segmentation, ignoring the abundant local structure that could be of utility even in the absence of fully connected structures. For example, in incremental (simultaneous) machine translation (Yarmohammadi et al., 2013), sub-sentential segments are translated independently and sequentially, hence the fully-connected syntactic structure is not generally available. Even so, locally-connected source language parse structures can inform both segmentation and translation of each segment in such a translation scenario.

One way to provide local hierarchical syntactic structures without fully connected trees is to focus on providing full hierarchical annotations for structures within a local window, ignoring global constituents outside that window. We follow the XML community in naming structures of this type *hedges* (not to be confused with the rhetorical device of the same name), due to the fact that they are like smaller versions of trees which occur in sequences. Such structures may be of utility to various structured inference tasks, as well as within a full parsing pipeline, to quickly constrain subsequent inference, much as finite-state models such as supertagging (Bangalore and Joshi, 1999) or chart cell constraints (Roark and Hollingshead, 2008; Roark et al., 2012) are used.

In this paper, we consider the problem of *hedge parsing*, i.e., discovering every constituent of length up to some span L. Similar constraints have been used in dependency parsing (Eisner and Smith, 2005; Dreyer et al., 2006), where the use of hard constraints on the distance between heads and dependents is known as vine parsing. It is also reminiscent of so-called Semi-Markov models (Sarawagi and Cohen, 2004), which allow finite-state models to reason about segments rather than just tags by imposing segment length limits. In the XML community, trees and hedges are used for models of XML document instances and for the contents of elements (Brüggemann-Klein and Wood, 2004). As far as we know, this paper is the first to consider this sort of partial parsing approach for natural language.

We pursue this topic via tree transformation, whereby non-root non-terminals labeling constituents of span $> L$ in the tree are recursively elided and their children promoted to attach to their parent. In such a way, hedges are sequentially connected to the top-most non-terminal in the tree, as demonstrated in Figure 1. After applying such a transform to a treebank, we can induce grammars and modify parsing to search as needed to recover just these constituents.

In this paper, we propose several methods to

Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Short Papers), pages 797–802,
Baltimore, Maryland, USA, June 23-25 2014.

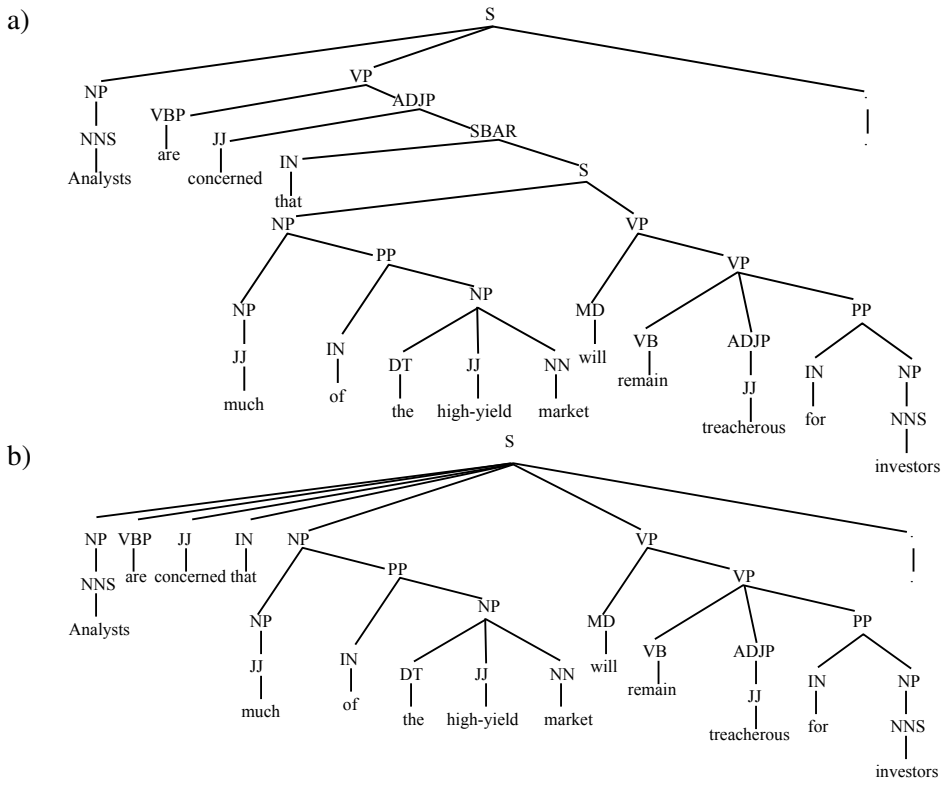


Figure 1: a) Full parse tree, b) Hedge parse tree with maximum constituent span of 7 ($L = 7$).

parse hedge constituents and examine their accuracy/efficiency tradeoffs. This is compared with a baseline of parsing with a typically induced context-free grammar and transforming the result via the hedge transform, which provides a ceiling on accuracy and a floor on efficiency. We investigate pre-segmenting the sentences with a finite-state model prior to hedge parsing, and achieve large speedups relative to hedge parsing the whole string, though at a loss in accuracy due to cascading segmentation errors. In all cases, we find it crucial that our "hedgebank" grammars be retrained to match the conditions during inference.

2 Methods

In this section, we present the details of our approach. First, we present the simple tree transform from a full treebank parse tree to a (root attached) sequence of hedges. Next, we discuss modifications to inference and the resulting computational complexity gains. Finally, we discuss segmenting to further reduce computational complexity.

2.1 Hedge Tree Transform

The hedge tree transform converts the original parse tree into a hedge parse tree. In the resulting hedge parse tree, every child of the top-most node spans at most L words. To transform an original tree to a hedge tree, we remove every non-terminal with span larger than L and attach its children to its parent. We label span length on each node by recursively summing the span lengths of each node's children, with terminal items by definition having span 1. A second top-down pass evaluates each node before evaluating its children, and removes nodes spanning $> L$ words. For example, the span of the non-root *S*, *SBAR*, *ADJP*, and *VP* nodes in Figure 1(a) have spans between 10 and 13, hence are removed in the tree in Figure 1(b).

If we apply this transform to an entire treebank, we can use the transformed trees to induce a PCFG for parsing. Figure 2 plots the percentage of constituents from the original WSJ Penn treebank (sections 2-21) retained in the transformed version, as we vary the maximum span length parameter L. Over half of constituents have span 3 or less (which includes frequent base noun phrases); $L = 7$ covers approximately three quarters of the original constituents, and $L = 15$ over 90%. Most experiments in this paper will focus on $L = 7$, which is short enough to provide a large speedup yet still cover a large fraction of constituents.

2.2 Hedge Parsing

As stated earlier, our brute-force baseline approach is to parse the sentence using a full context-free grammar (CFG) and then hedge-transform the result. This method should yield a ceiling on

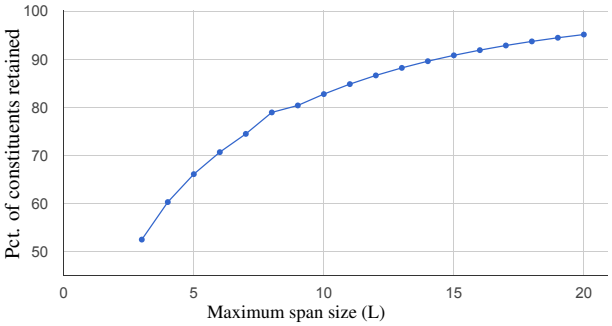


Figure 2: Percentage of constituents retained at various span length parameters

hedge-parsing accuracy, as it has access to rich contextual information (as compared to grammars trained on transformed trees). Naturally, inference will be slow; we aim to improve efficiency upon this baseline while minimizing accuracy loss.

Since we limit the span of non-terminal labels, we can constrain the search performed by the parser, greatly reduce the CYK processing time. In essence, we perform no work in chart cells spanning more than L words, except for the cells along the periphery of the chart, which are just used to connect the hedges to the root. Consider the flat tree in Figure 1(b). For use by a CYK parsing algorithm, trees are binarized prior to grammar induction, resulting in special non-terminals created by binarization. Other than the symbol at the root of the tree, the only constituents with span length greater than L in the binarized tree will be labeled with these special binarization non-terminals. Further, if the binarization systematically groups the leftmost or the rightmost children under these new non-terminals (the most common strategy), then constituents with span greater than L will either begin at the first word (leftmost grouping) or end at the last word (rightmost), further constraining the number of cells in the chart requiring work.

Complexity of parsing with a full CYK parser is $O(n^3|G|)$ where n is the length of input and $|G|$ is the grammar size constant. In contrast, complexity of parsing with a hedge constrained CYK is reduced to $O((nL^2+n^2)|G|)$. To see that this is the case, consider that there are $O(nL)$ cells of span L or less, and each has a maximum of L midpoints, which accounts for the first term. Beyond these, there are $O(n)$ remaining active cells with $O(n)$ possible midpoints, which accounts for the second term. Note also that these latter cells (spanning $> L$ words) may be less expensive, as the set of possible non-terminals is reduced to only those introduced by binarization.

It is possible to parse with a standardly induced PCFG using this sort of hedge constrained parsing that only considers a subset of the chart cells, and speedups are achieved, however this is clearly non-optimal, since the model is ill-suited to combining hedges into flat structures at the root of the tree. Space constraints preclude inclusion of trials with this method, but the net result is a severe degradation in accuracy (tens of points of F-measure) versus standard parsing. Thus, we train a grammar in a matched condition, which we call it a *hedgebank grammar*. A hedgebank grammar is a fully functional PCFG which is learned from a hedge transformed treebank. A hedgebank grammar can be used with any standard parsing algorithm, i.e., these are not generally finite-state equivalent models. However, using the Berkeley grammar learner (see §3), we find that hedgebank grammars are typically smaller than treebank grammars, reducing the grammar constant and contributing to faster inference.

A unique property of hedge constituents compared to constituents in the original parse trees is that they are sequentially connected to the topmost node. This property enables us to chunk the sentence into segments that correspond to complete hedges, and parse the segments independently (and simultaneously) instead of parsing the entire sentence. In section 2.3, we present our approach to hedge segmentation.

In all scenarios where the chart is constrained to search for hedges, we learn a hedgebank grammar, which is matched to the maximum length allowed by the parser. In the pre-segmentation scenario, we first decompose the hedge transformed treebank into its hedge segments and then learn a hedgebank grammar from the new corpus.

2.3 Hedge Segmentation

In this section we present our segmentation model which takes the input sentence and chunks it into appropriate segments for hedge parsing. We treat this as a binary classification task which decides if a word can begin a new hedge. We use hedge segmentation as a finite-state pre-processing step for hedge context-free parsing.

Our task is to learn which words can begin (B) a hedge constituent. Given a set of labeled pairs (S, H) where S is a sentence of n words $w_1 \ldots w_n$ and H is its hedge parse tree, word w_b belongs to B if there is a hedge constituent spanning $w_b \ldots w_e$ for some $e \geq b$ and w_b belongs to $\bar{B}$ otherwise. To predict the hedge boundaries more accurately, we grouped consecutive unary or POS-

tag hedges together under a new non-terminal labeled G. Unlabeled segmentation tags for the words in the example sentence in Figure 1(b) are:

"Analysts/B are/$\bar{B}$ concerned/$\bar{B}$ that/$\bar{B}$ much/B of/$\bar{B}$ the/$\bar{B}$ high-yield/$\bar{B}$ market/$\bar{B}$ will/B remain/$\bar{B}$ treacherous/$\bar{B}$ for/$\bar{B}$ investors/$\bar{B}$ ./B"

In addition to the simple unlabeled segmentation with B and $\bar{B}$ tags, we try a labeled segmentation with B_C and $\bar{B}_C$ tags where C is hedge constituent type. We restrict the types to the most important types – following the 11 chunk types annotated in the CoNLL-2000 chunking task (Sang and Buchholz, 2000) – by replacing all other types with a new type *OUT*. Thus, "Analysts" is labeled B_G; "much", B_{NP}; "will", B_{VP} and so on.

To automatically predict the class of each word position, we train a multi-class classifier from labeled training data using a discriminative linear model, learning the model parameters with the averaged perceptron algorithm (Collins, 2002). We follow Roark et al. (2012) in the features they used to label words as beginning or ending constituents. The segmenter extracts features from word and POS-tag input sequences and hedge-boundary tag output sequences. The feature set includes trigrams of surrounding words, trigrams of surrounding POS tags, and hedge-boundary tags of the previous words. An additional orthographical feature set is used to tag rare[1] and unknown words. This feature set includes prefixes and suffixes of the words (up to 4 characters), and presence of a hyphen, digit, or an upper-case character. Reported results are for a Markov order-2 segmenter, which includes features with the output classes of the previous two words.

3 Experimental Results

We ran all experiments on the WSJ Penn Treebank corpus (Marcus et al., 1999) using section 2-21 for training, section 24 for development, and section 23 for testing. We performed exhaustive CYK parsing using the BUBS parser[2] (Bodenstab et al., 2011) with Berkeley SM6 latent-variable grammars (Petrov and Klein, 2007) learned by the Berkeley grammar trainer with default settings. We compute accuracy from the 1-best Viterbi tree extracted from the chart using the standard EVALB script. Accuracy results are reported as precision, recall and F1-score, the harmonic mean between the two. In all trials, we evaluate accuracy with respect to the hedge transformed reference treebank, i.e., we are not penalizing the parser for not discovering constituents longer than the maximum length. Segmentation accuracy is reported as an F1-score of unlabeled segment bracketing. We ran timing tests on an Intel 2.66GHz processor with 3MB of cache and 2GB of memory. Note that segmentation time is negligible compared to the parsing time, hence is omitted in reported time. Efficiency results are reported as number of words parsed per second (w/s).

	Hedge Parsing Acc/Eff			
Parser	P	R	F1	w/s
Full w/full CYK	88.8	89.2	89.0	2.4
Hedgebank	87.6	84.4	86.0	25.7

Table 1: Hedge parsing results on section 24 for $L = 7$.

Table 1 presents hedge parsing accuracy on the development set for the full parsing baseline, where the output of regular PCFG parsing is transformed to hedges and evaluated, versus parsing with a hedgebank grammar, with no segmentation of the strings. We find an order of magnitude speedup of parsing, but at the cost of 3 percent F-measure absolute. Note that most of that loss is in recall, indicating that hedges predicted in that condition are nearly as reliable as in full parsing.

Table 2 shows the results on the development set when segmenting prior to hedge parsing. The first row shows the result with no segmentation, the same as the last row in Table 1 for ease of reference. The next row shows behavior with perfect segmentation. The final two rows show performance with automatic segmentation, using a model that includes either unlabeled or labeled segmentation tags, as described in the last section. Segmentation accuracy is better for the model with labels, although overall that accuracy is rather low. We achieve nearly another order of magnitude speedup over hedge parsing without segmentation, but again at the cost of nearly 5 percent F1.

Table 3 presents results of our best configurations on the eval set, section 23. The results show the same patterns as on the development set. Finally, Figure 3 shows the speed of inference, la-

Table 2: Hedge segmentation and parsing results on section 24 for $L = 7$.

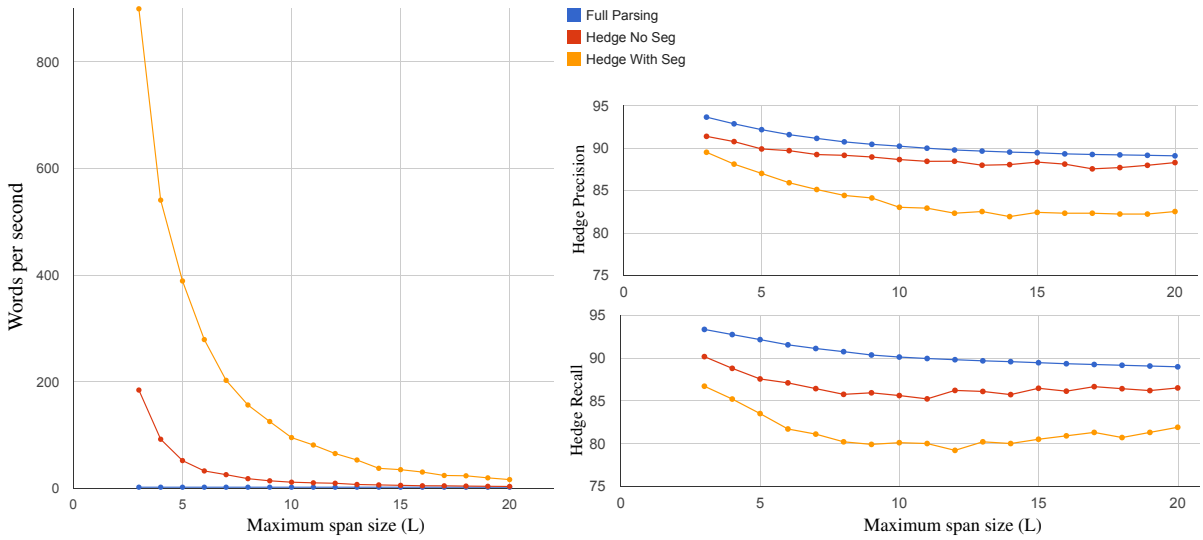
Segmentation	Seg F1	Hedge Parsing Acc/Eff P	R	F1	w/s
None	n/a	87.6	84.4	86.0	25.7
Oracle	100	91.3	88.9	90.1	188.6
Unlabeled	80.6	77.2	75.3	76.2	159.1
Labeled	83.8	83.1	79.5	81.3	195.8

[1] Rare words occur less than 5 times in the training data.

[2] https://code.google.com/p/bubs-parser

Segmentation	Grammar	Segmentation Acc			Hedge Parsing Acc/Eff			
		P	R	F1	P	R	F1	w/s
None	Full w/full CYK	n/a			90.3	90.3	90.3	2.7
None	Hedgebank	n/a			88.3	85.3	86.8	26.2
Labeled	Hedgebank	84.0	86.6	85.3	85.1	81.1	83.0	203.0

Table 3: Hedge segmentation and parsing results on test data, section 23, for $L = 7$.



a) b)

Figure 3: Hedge parsing a) efficiency, and b) accuracy on test data, section 23, for $L = 3$–20.

beled precision and labeled recall of annotating hedge constituents on the test set as a function of the maximum span parameter L, versus the baseline parser. Keep in mind that the number of reference constituents increases as L increases, hence both precision and recall can decrease as the parameter grows. Segmentation achieves large speedups for smaller L values, but the accuracy degradation is consistent, pointing to the need for improved segmentation.

4 Conclusion and Future Work

We proposed a novel partial parsing approach for applications that require a fast syntactic analysis of the input beyond shallow bracketing. The span-limit parameter allows tuning the annotation of internal structure as appropriate for the application domain, trading off annotation complexity against inference time. These properties make hedge parsing potentially very useful for incremental text or speech processing, such as streaming text analysis or simultaneous translation.

One interesting characteristic of these annotations is that they allow for string segmentation prior to inference, provided that the segment boundaries do not cross any hedge boundaries. We found that baseline segmentation models did provide a significant speedup in parsing, but that cascading errors remain a problem.

There are many directions of future work to pursue here. First, the current results are all for exhaustive CYK parsing, and we plan to perform a detailed investigation of the performance of hedgebank parsing with prioritization and pruning methods of the sort available in BUBS (Bodenstab et al., 2011). Further, this sort of annotation seems well suited to incremental parsing with beam search, which has been shown to achieve high accuracies even for fully connected parsing (Zhang and Clark, 2011). Improvements to the transform (e.g., grouping items not in hedges under non-terminals) and to the segmentation model (e.g., increasing precision at the expense of recall) could improve accuracy without greatly reducing efficiency. Finally, we intend to perform an extrinsic evaluation of this parsing in an on-line task such as simultaneous translation.

Acknowledgments

This work was supported in part by NSF grant #IIS-0964102. Any opinions, findings, conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the NSF.

References

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

Nathan Bodenstab, Aaron Dunlop, Keith Hall, and Brian Roark. 2011. Beam-width prediction for efficient context-free parsing. In *Proceedings of the 49th Annual Meeting ACL: HLT*, pages 440–449.

Anne Brüggemann-Klein and Derick Wood. 2004. Balanced context-free grammars, hedge grammars and pushdown caterpillar automata. In *Extreme Markup Languages*.

Michael Collins. 2002. Discriminative training methods for hidden Markov models: theory and experiments with perceptron algorithms. In *Proceedings of the conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1–8.

Markus Dreyer, David A. Smith, and Noah A. Smith. 2006. Vine parsing and minimum risk reranking for speed and precision. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL)*, pages 201–205.

Jason Eisner and Noah A. Smith. 2005. Parsing with soft and hard constraints on dependency length. In *Proceedings of the Ninth International Workshop on Parsing Technology (IWPT)*, pages 30–41.

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*, pages 1663–1666.

Brian Roark and Kristy Hollingshead. 2008. Classifying chart cells for quadratic complexity context-free inference. In *Proceedings of the 22nd International Conference on Computational Linguistics*, pages 745–751.

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.

Erik F. Tjong Kim Sang and Sabine Buchholz. 2000. Introduction to the CoNLL-2000 shared task: Chunking. In *Proceedings of Conference on Computational Natural Language Learning (CoNLL)*, pages 127–132.

Sunita Sarawagi and William W. Cohen. 2004. Semi-Markov conditional random fields for information extraction. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1185–1192.

Mahsa Yarmohammadi, Vivek K. Rangarajan Sridhar, Srinivas Bangalore, and Baskaran Sankaran. 2013. Incremental segmentation and decoding strategies for simultaneous translation. In *Proceedings of the 6th International Joint Conference on Natural Language Processing (IJCNLP)*, pages 1032–1036.

Yue Zhang and Stephen Clark. 2011. Syntactic processing using the generalized perceptron and beam search. *Computational Linguistics*, 37(1):105–151.