Parser Combination by Reparsing

Kenji Sagae and **Alon Lavie**
Language Technologies Institute
Carnegie Mellon University
Pittsburgh, PA 15213
{sagae,alavie@cs.cmu.edu}

Abstract

We present a novel parser combination scheme that works by reparsing input sentences once they have already been parsed by several different parsers. We apply this idea to dependency and constituent parsing, generating results that surpass state-of-the-art accuracy levels for individual parsers.

1 Introduction

Over the past decade, remarkable progress has been made in data-driven parsing. Much of this work has been fueled by the availability of large corpora annotated with syntactic structures, especially the Penn Treebank (Marcus et al., 1993). In fact, years of extensive research on training and testing parsers on the Wall Street Journal (WSJ) corpus of the Penn Treebank have resulted in the availability of several high-accuracy parsers.

We present a framework for combining the output of several different accurate parsers to produce results that are superior to those of each of the individual parsers. This is done in a two stage process of *reparsing*. In the first stage, *m* different parsers analyze an input sentence, each producing a syntactic structure. In the second stage, a parsing algorithm is applied to the original sentence, taking into account the analyses produced by each parser in the first stage. Our approach produces results with accuracy above those of the best individual parsers on both dependency and constituent parsing of the standard WSJ test set.

2 Dependency Reparsing

In dependency reparsing we focus on unlabeled dependencies, as described by Eisner (1996). In this scheme, the syntactic structure for a sentence with *n* words is a dependency tree representing head-dependent relations between pairs of words.

When *m* parsers each output a set of dependencies (forming *m* dependency structures) for a given sentence containing *n* words, the dependencies can be combined in a simple word-by-word voting scheme, where each parser votes for the head of each of the *n* words in the sentence, and the head with most votes is assigned to each word. This very simple scheme guarantees that the final set of dependencies will have as many votes as possible, but it does not guarantee that the final voted set of dependencies will be a well-formed dependency tree. In fact, the resulting graph may not even be connected. Zeman & Žabokrtský (2005) apply this dependency voting scheme to Czech with very strong results. However, when the constraint that structures must be well-formed is enforced, the accuracy of their results drops sharply.

Instead, if we reparse the sentence based on the output of the *m* parsers, we can maximize the number of votes for a well-formed dependency structure. Once we have obtained the *m* initial dependency structures to be combined, the first step is to build a graph where each word in the sentence is a node. We then create weighted directed edges between the nodes corresponding to words for which dependencies are obtained from each of the initial structures.[1] In cases where more than one dependency structure indicates that an edge should be created, the corresponding weights are simply added. As long as at least one of the *m* initial structures is a well-formed dependency structure, the directed graph created this way will be connected.

[1] Determining the weights is discussed in section 4.1.

Once this graph is created, we reparse the sentence using a dependency parsing algorithm such as, for example, one of the algorithms described by McDonald et al. (2005). Finding the optimal dependency structure given the set of weighted dependencies is simply a matter of finding the maximum spanning tree (MST) for the directed weighted graph, which can be done using the Chu-Liu/Edmonds directed MST algorithm (Chu & Liu, 1965; Edmonds, 1967). The maximum spanning tree maximizes the votes for dependencies given the constraint that the resulting structure must be a tree. If projectivity (no crossing branches) is desired, Eisner's (1996) dynamic programming algorithm (similar to CYK) for dependency parsing can be used instead.

3 Constituent Reparsing

In constituent reparsing we deal with labeled constituent trees, or phrase structure trees, such as those in the Penn Treebank (after removing traces, empty nodes and function tags). The general idea is the same as with dependencies. First, *m* parsers each produce one parse tree for an input sentence. We then use these *m* initial parse trees to guide the application of a parse algorithm to the input.

Instead of building a graph out of words (nodes) and dependencies (edges), in constituent reparsing we use the *m* initial trees to build a weighted parse chart. We start by decomposing each tree into its constituents, with each constituent being a 4-tuple [*label, begin, end, weight*], where *label* is the phrase structure type, such as NP or VP, *begin* is the index of the word where the constituent starts, *end* is the index of the word where the constituent ends plus one, and *weight* is the weight of the constituent. As with dependencies, in the simplest case the weight of each constituent is simply 1.0, but different weighting schemes can be used. Once the initial trees have been broken down into constituents, we put all the constituents from all of the *m* trees into a single list. We then look for each pair of constituents *A* and *B* where the *label*, *begin*, and *end* are identical, and merge *A* and *B* into a single constituent with the same *label*, *begin*, and *end*, and with *weight* equal to the *weight* of *A* plus the *weight* of *B*. Once no more constituent mergers are possible, the resulting constituents are placed on a standard parse chart, but where the constituents in the chart do not contain back-pointers indicating what smaller constituents they contain. Building the final tree amounts to determining these back-pointers. This can be done by running a bottom-up chart parsing algorithm (Allen, 1995) for a weighted grammar, but instead of using a grammar to determine what constituents can be built and what their weights are, we simply constrain the building of constituents to what is already in the chart (adding the weights of constituents when they are combined). This way, we perform an exhaustive search for the tree that represents the heaviest combination of constituents that spans the entire sentence as a well-formed tree.

A problem with simply considering all constituents and picking the heaviest tree is that this favors recall over precision. Balancing precision and recall is accomplished by discarding every constituent with weight below a threshold *t* before the search for the final parse tree starts. In the simple case where each constituent starts out with weight 1.0 (before any merging), this means that a constituent is only considered for inclusion in the final parse tree if it appears in at least *t* of the *m* initial parse trees. Intuitively, this should increase precision, since we expect that a constituent that appears in the output of more parsers to be more likely to be correct. By changing the threshold *t* we can control the precision/recall tradeoff.

Henderson and Brill (1999) proposed two parser combination schemes, one that picks an entire tree from one of the parsers, and one that, like ours, builds a new tree from constituents from the initial trees. The latter scheme performed better, producing remarkable results despite its simplicity. The combination is done with a simple majority vote of whether or not constituents should appear in the combined tree. In other words, if a constituent appears at least $(m + 1)/2$ times in the output of the *m* parsers, the constituent is added to the final tree. This simple vote resulted in trees with f-score significantly higher than the one of the best parser in the combination. However, the scheme heavily favors precision over recall. Their results on WSJ section 23 were 92.1 precision and 89.2 recall (90.61 f-score), well above the most accurate parser in their experiments (88.6 f-score).

4 Experiments

In our dependency parsing experiments we used unlabeled dependencies extracted from the Penn

Treebank using the same head-table as Yamada and Matsumoto (2003), using sections 02-21 as training data and section 23 as test data, following (McDonald et al., 2005; Nivre & Scholz, 2004; Yamada & Matsumoto, 2003). Dependencies extracted from section 00 were used as held-out data, and section 22 was used as additional development data. For constituent parsing, we used the section splits of the Penn Treebank as described above, as has become standard in statistical parsing research.

4.1 Dependency Reparsing Experiments

Six dependency parsers were used in our combination experiments, as described below.

The deterministic shift-reduce parsing algorithm of (Nivre & Scholz, 2004) was used to create two parsers[2], one that processes the input sentence from left-to-right (LR), and one that goes from right-to-left (RL). Because this deterministic algorithm makes a single pass over the input string with no back-tracking, making decisions based on the parser's state and history, the order in which input tokens are considered affects the result. Therefore, we achieve additional parser diversity with the same algorithm, simply by varying the direction of parsing. We refer to the two parsers as LR and RL.

The deterministic parser of Yamada and Matsumoto (2003) uses an algorithm similar to Nivre and Scholz's, but it makes several successive left-to-right passes over the input instead of keeping a stack. To increase parser diversity, we used a version of Yamada and Matsumoto's algorithm where the direction of each of the consecutive passes over the input string alternates from left-to-right and right-to-left. We refer to this parser as LRRL.

The large-margin parser described in (McDonald et al., 2005) was used with no alterations. Unlike the deterministic parsers above, this parser uses a dynamic programming algorithm (Eisner, 1996) to determine the best tree, so there is no difference between presenting the input from left-to-right or right-to-left.

Three different weight configurations were considered: (1) giving all dependencies the same weight; (2) giving dependencies different weights, depending only on which parser generated the dependency; and (3) giving dependencies different weights, depending on which parser generated the dependency, and the part-of-speech of the dependent word. Option 2 takes into consideration that parsers may have different levels of accuracy, and dependencies proposed by more accurate parsers should be counted more heavily. Option 3 goes a step further, attempting to capitalize on the specific strengths of the different parsers.

The weights in option 2 are determined by computing the accuracy of each parser on the held-out set (WSJ section 00). The weights are simply the corresponding parser's accuracy (number of correct dependencies divided by the total number of dependencies). The weights in option 3 are determined in a similar manner, but different accuracy figures are computed for each part-of-speech.

Table 1 shows the dependency accuracy and root accuracy (number of times the root of the dependency tree was identified correctly divided by the number of sentences) for each of the parsers, and for each of the different weight settings in the reparsing experiments (numbered according to their descriptions above).

System	Accuracy	Root Acc.
LR	91.0	92.6
RL	90.1	86.3
LRRL	89.6	89.1
McDonald	90.9	94.2
Reparse dep 1	91.8	96.0
Reparse dep 2	92.1	95.9
Reparse dep 3	**92.7**	**96.6**

Table 1: Dependency accuracy and root accuracy of individual dependency parsers and their combination under three different weighted reparsing settings.

4.2 Constituent Reparsing Experiments

The parsers that were used in the constituent reparsing experiments are: (1) Charniak and Johnson's (2005) reranking parser; (2) Henderson's (2004) synchronous neural network parser; (3) Bikel's (2002) implementation of the Collins (1999) model 2 parser; and (4) two versions of Sagae and Lavie's (2005) shift-reduce parser, one using a maximum entropy classifier, and one using support vector machines.

Henderson and Brill's voting scheme mentioned in section 3 can be emulated by our reparsing approach by setting all weights to 1.0 and t to $(m + 1)/2$, but better results can be obtained by setting appropriate weights and adjusting the precision/recall tradeoff. Weights for different types of

[2] Nivre and Scholz use memory based learning in their experiments. Our implementation of their parser uses support vector machines, with improved results.

constituents from each parser can be set in a similar way to configuration 3 in the dependency experiments. However, instead of measuring accuracy for each part-of-speech tag of dependents, we measure precision for each non-terminal label.

The parameter *t* is set using held-out data (from WSJ section 22) and a simple hill-climbing procedure. First we set *t* to *(m + 1)/2* (which heavily favors precision). We then repeatedly evaluate the combination of parsers, each time decreasing the value of *t* (by 0.01, say). We record the values of *t* for which precision and recall were closest, and for which f-score was highest.

Table 2 shows the accuracy of each individual parser and for three reparsing settings. Setting 1 is the emulation of Henderson and Brill's voting. In setting 2, *t* is set for balancing precision and recall. In setting 3, *t* is set for highest f-score.

System	Precision	Recall	F-score
Charniak/Johnson	91.3	90.6	91.0
Henderson	90.2	89.1	89.6
Bikel (Collins)	88.3	88.1	88.2
Sagae/Lavie (a)	86.9	86.6	86.7
Sagae/Lavie (b)	88.0	87.8	87.9
Reparse 1	**95.1**	88.5	91.6
Reparse 2	91.8	**91.9**	91.8
Reparse 3	93.2	91.0	**92.1**

Table 2: Precision, recall and f-score of each constituent parser and their combination under three different reparsing settings.

5 Discussion

We have presented a reparsing scheme that produces results with accuracy higher than the best individual parsers available by combining their results. We have shown that in the case of dependencies, the reparsing approach successfully addresses the issue of constructing high-accuracy well-formed structures from the output of several parsers. In constituent reparsing, held-out data can be used for setting a parameter that allows for balancing precision and recall, or increasing f-score. By combining several parsers with f-scores ranging from 91.0% to 86.7%, we obtain reparsed results with a 92.1% f-score.

References

Allen, J. (1995). *Natural Language Understanding* (2nd ed.). Redwood City, CA: The Benjamin/Cummings Publishing Company, Inc.

Bikel, D. (2002). Design of a multi-lingual, parallel-processing statistical parsing engine. In *Proceedings of HLT2002*. San Diego, CA.

Charniak, E., & Johnson, M. (2005). Coarse-to-fine n-best parsing and MaxEnt discriminative reranking. In *Proceedings of the 43rd meeting of the Association for Computational Linguistics*. Ann Arbor, MI.

Chu, Y. J., & Liu, T. H. (1965). On the shortest arborescence of a directed graph. *Science Sinica*(14), 1396-1400.

Edmonds, J. (1967). Optimum branchings. *Journal of Research of the National Bureau of Standards*(71B), 233-240.

Eisner, J. (1996). Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the International Conference on Computational Linguistics (COLING'96)*. Copenhagen, Denmark.

Henderson, J. (2004). Discriminative training of a neural network statistical parser. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics*. Barcelona, Spain.

Henderson, J., & Brill, E. (1999). Exploiting diversity in natural language processing: combining parsers. In *Proceedings of the Fourth Conference on Empirical Methods in Natural Language Processing (EMNLP).*

Marcus, M. P., Santorini, B., & Marcinkiewics, M. A. (1993). Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics, 19.*

McDonald, R., Pereira, F., Ribarov, K., & Hajic, J. (2005). Non-Projective Dependency Parsing using Spanning Tree Algorithms. In *Proceedings of the Conference on Human Language Technologies/Empirical Methods in Natural Language Processing (HLT-EMNLP)*. Vancouver, Canada.

Nivre, J., & Scholz, M. (2004). Deterministic dependency parsing of English text. In *Proceedings of the 20th International Conference on Computational Linguistics* (pp. 64-70). Geneva, Switzerland.

Sagae, K., & Lavie, A. (2005). A classifier-based parser with linear run-time complexity. In *Proceedings of the Ninth International Workshop on Parsing Technologies.* Vancouver, Canada.

Yamada, H., & Matsumoto, Y. (2003). Statistical dependency analysis using support vector machines. In *Proceedings of the Eighth International Workshop on Parsing Technologies*. Nancy, France.

Zeman, D., & Žabokrtský, Z. (2005). Improving Parsing Accuracy by Combining Diverse Dependency Parsers. In *Proceedings of the International Workshop on Parsing Technologies*. Vancouver, Canada.