

# Parsing to Stanford Dependencies: Trade-offs between speed and accuracy

Daniel Cer<sup>\*</sup>, Marie-Catherine de Marneffe<sup>‡</sup>, Daniel Jurafsky<sup>‡</sup> and Christopher D. Manning<sup>\*‡</sup>

<sup>\*</sup>Computer Science Department and <sup>‡</sup>Linguistics Department  
Stanford University  
Stanford, CA 94305, USA

## Abstract

Recent years have seen an increase in the use of dependency representations throughout various natural language processing (NLP) tasks. The Stanford dependency scheme (de Marneffe et al., 2006) in particular has gained popularity: it is widely used in both the NLP community (i.a., Blake (2007), Banko et al. (2007), Kessler (2008)) and the biomedical text mining community (i.a., Pyysalo et al. (2007), Björne et al. (2009), Van Landeghem et al. (2009)). Stanford Dependencies have traditionally been extracted from constituent parses. However, using the default configuration of off-the-self constituent parsers, it is quite slow to obtain dependencies from raw text as the production of parse trees is very time consuming.

In this paper, we systematically explore different methods for obtaining Stanford Dependencies. There has been some work examining accuracy using different constituent parsers to generate Stanford Dependencies (Clegg and Shepherd, 2007). Miyao et al. (2008) developed the approach of automatically converting parsers' default output into dependency representations to allow evaluation of the contribution of the parser and the representation on a relation extraction task. We expand the investigation by looking at time and accuracy trade-offs and examining how such constituent parsers compare to fast algorithms that have been specifically developed for dependency parsing, such as those found in the MaltParser package (Nivre et al., 2006) and MSTParser (McDonald et al., 2005). We then compare these dependency parsers with techniques for speeding up the traditional extraction pipeline, namely more aggressive pruning in constituent parsers. We contrast the different approaches in terms of aggregate speed and accuracy and provide a detailed analysis of characteristic errors of each.

We compare popular state-of-the-art constituent parsers: Stanford englishPCFG v1.6.1 (Klein and Manning, 2003), Charniak 05Aug16 (Charniak, 2000), Charniak-Johnson June06 (CJ) (Charniak and Johnson, 2005), and Bikel v1.2 (Bikel, 2004). Such

parsers differ in terms of accuracy, speed and the options they provide to trade off time with accuracy. We also compare different dependency parsers: several models from the MaltParser package v1.3 (Nivre, Nivre eager, and Covington) (Nivre et al., 2006), a minimum-spanning tree parser (MSTParser 0.4.3b) (McDonald et al., 2005), and the rule-based ReLEx parser 1.2.0 (Ross and Vepstas, 2008). The ReLEx parser supports a Stanford dependency compatibility mode. For the others, we train their models on the Stanford basic dependencies. The basic dependencies provide projective grammatical relations between every word in the sentence, without any collapsing or propagation of dependencies (de Marneffe and Manning, 2008). The resulting dependency trees can then be systematically transformed into the standard Stanford dependency representation.

Directly training non-projective parsing models, such as MSTParser or Covington, on the standard Stanford dependency representation is not advisable since that representation is not just non-projective but the semantic graphs it defines do not strictly follow a tree structure.

Table 1 reports attachment accuracy for the different parsers on section 22 of the Penn TreeBank. "Gold" dependencies were obtained by running the Stanford extraction code on the gold phrase structure trees. As in previous work, the automatic conversion of gold standard parse trees to dependencies has not been manually checked. The table also gives the time taken to generate the dependencies. The dependency parsers require that the data is part-of-speech tagged. We use the Stanford POS tagger v1.6 (Toutanova et al., 2003). Additional gains in speed could be obtained by using a faster tagger.

The fastest parsers are those included in the Malt package, Nivre, Nivre eager, and Covington, when interactions between model features are not used.<sup>1</sup> The

<sup>1</sup>As released, the MaltParser (v1.3 and earlier) has a bug that causes parse time to be quadratic in *the number of*

| Type        | Parser                 | Attachment acc. |              | Time    |             |                 |             |
|-------------|------------------------|-----------------|--------------|---------|-------------|-----------------|-------------|
|             |                        | Unlabeled       | Labeled      | POS tag | Parse       | Dep. extraction | Total       |
| Constituent | Stanford               | 90.06           | 87.19        | –       | 26:36       | 1:39            | 28:15       |
|             | Charniak               | 92.78           | 90.09        | –       | 20:54       | 1:40            | 22:34       |
|             | CJ                     | <b>93.72</b>    | <b>91.15</b> | –       | 21:03       | 1:44            | 22:47       |
|             | Bikel                  | 91.43           | 88.12        | –       | 53:58       | 1:39            | 55:37       |
| Dependency  | Covington              | 84.99           | 82.11        | 2:43    | <b>0:15</b> | –               | <b>2:58</b> |
|             | Nivre Eager            | 84.96           | 81.82        | 2:43    | <b>0:14</b> | –               | <b>2:57</b> |
|             | Nivre                  | 85.25           | 82.08        | 2:43    | <b>0:12</b> | –               | <b>2:55</b> |
|             | Nivre Feature Interact | 89.62           | 86.69        | 2:43    | 4:34        | –               | 7:17        |
|             | MSTParser              | 90.18           | 86.75        | 2:43    | 11:58       | –               | 14:55       |
|             | RelEx                  | 63.36           | 55.54        | –       | 40:57       | –               | 40:57       |

Table 1: Unlabeled and labeled attachment accuracy (%) and time (min:seconds) to generate Stanford Dependencies with different types of parsers (constituent vs. dependency). When applicable, dependency extraction times are given for the Stanford basic dependencies. Converting from the Stanford basic dependencies to the final representation took an additional 4 to 5 seconds per parser.

| Parser       | Attachment acc. |         | Time    |       |                 |       |
|--------------|-----------------|---------|---------|-------|-----------------|-------|
|              | Unlabeled       | Labeled | POS tag | Parse | Dep. extraction | Total |
| Charniak T10 | 84.10           | 80.27   | –       | 2:23  | 1:37            | 2:00  |
| Charniak T50 | 92.10           | 89.31   | –       | 3:23  | 1:37            | 5:00  |
| CJ T10       | 85.16           | 81.47   | –       | 2:41  | 1:35            | 4:16  |
| CJ T50       | 92.99           | 90.27   | –       | 5:08  | 1:38            | 6:46  |

Table 2: Unlabeled and labeled attachment accuracy (%) and time (min:seconds) to generate Stanford Dependencies with different beams of the Charniak and Charniak-Johnson parsers.

dependencies extracted from the constituent parsers are the most accurate, but they are also the slowest to generate. The MSTParser is nearly as accurate as the constituent parsers but is 34% or more faster. It is, however, twice as slow as and only modestly more accurate than the Nivre parser when using feature interactions.<sup>2</sup>

Both the Charniak and the CJ parsers allow users to

*words in the corpus being parsed* due to pre-insertion in a list that grows with each parsing prediction made. We fixed this problem yielding a patched version which is about 2 orders of magnitude faster than the latest release of the package on the data sets reported here when feature interactions are not used. The speed of the MaltParser is also significantly impacted by the large number of feature dot products required (one for each support vector) when feature interactions are modeled using a SVM with a non-linear kernel. We thus modified the code so that a polynomial kernel can be simulated using a linear model. Doing so resulted in an approximately 5x speedup for our feature interaction results. Table 1 reports results after these fixes have been applied.

<sup>2</sup>We obtained similar speed and accuracy results for the other parsing algorithms included in the Malt package (Nivre eager and Covington).

trade off parsing accuracy for speed by adjusting how liberal the system is about expanding edges after the best-first-search has found one complete parse of the sentence: they constrain themselves to only examine  $T_{val}/10$  times more edges in search of a better parse. As seen in table 2, by adjusting this parameter down, the time required by these parsers is reduced to nearly that required by the fastest dependency parsing algorithms. In fact, of all the parsers we evaluate, Charniak T10 provides the fastest end-to-end option for obtaining Stanford Dependencies, since, unlike some other alternatives, it does not require the data to be already labeled with POS tags. Unfortunately, these gains come with a sizable reduction in dependency accuracy. However, by expanding the space of hypotheses explored by the parser, Charniak T50 achieves very competitive parsing accuracy. By also reranking the parse trees, CJ T50 is more accurate than nearly all other configurations, while requiring less time than all but the fastest specialized dependency parsers.

We performed error analysis on section 22 of the Penn TreeBank, the same data used for table 1. The low accuracy of RelEx is largely due to the parser omitting a

sizable number of dependencies. All the errors made by the constituent parsers are due to incorrect phrase structures leading to higher or lower attachment as well as to the use of the imprecise generic *dep* relation. Not surprisingly most of the errors occur with structures which are inherently hard to attach: subordinate clauses, prepositional and adverbial phrases. For example, in (1) *But the RTC also requires working capital to maintain the bad assets-1 of thrifts that are sold-1 until the assets-2 can be sold-2 separately.*, both Stanford and Charniak produce *xcomp*(requires, maintain) instead of *infmod*(capital, maintain). Stanford also misattaches the adverbial clause: *advcl*(sold-1, sold-2) instead of *advcl*(maintain, sold-2). Decreasing the beam size for the Charniak parser leads to a greater number of such errors, especially for embedded clauses (*ccomp*), relative clauses, prepositional phrases as well as auxiliaries.

Nivre and Nivre eager often produce more local attachments. For example, in (2) *The bill would prevent the Resolution Trust Corp. from raising temporary working capital by having an RTC-owned bank or thrift issue debt.*, we get *prepc\_by*(raising, having) instead of *prepc\_by*(prevent, having). Incorrect higher attachments sometimes occur, probably due to a lexical preference: in example (1), we have *rmod*(thrifts, sold-1) instead of *rmod*(assets-1, sold-1). A systematic error can be seen in the treatment of copulas. In most copular sentences, the Stanford Dependencies take the complement of the copular verb as the root. However, these parsers rarely gives such output, presumably because locally the attachment to the copula appears to be reasonable and being deterministic the parser is never able to back out of this decision.

In fact, most systematic errors made by such parsers can be attributed to their deterministic nature: once it mistakenly attaches a dependent that looks good given the local context, it cannot backtrack if it comes across a better candidate. To this extent, lexical preference can also conspire with locality and introduce parse errors. Even though the MSTParser is performing exact inference over all possible parses, it still makes some errors similar to those made by the deterministic parsers involving inappropriate local attachment. In this case, these errors are likely due to the feature set used by the MSTParser which favors local dependencies.

Notwithstanding the very large amount of research that has gone into dependency parsing algorithms in the last five years, our central conclusion is that the quality of the Charniak-Johnson parser is so high that in the vast majority of cases, dependency parse users are better off using it, and then converting the output to

dependencies. For small scale tasks, the CJ reranking parser is best due to its high level of accuracy. If parsing a larger corpus, the best choice is to still use CJ but to reduce the number of candidate parses explored by the algorithm. Interestingly enough, this option is both faster and more accurate than some of the special purpose dependency parsers. If parsing a massive corpus, and speed is crucial, our results suggest that the best choice is to use any one of the parsers included in the Malt package with a fast POS tagger.

## 1. References

- Michele Banko, Michael J. Cafarella, Stephen Soderland, Matt Broadhead, and Oren Etzioni. 2007. Open information extraction from the web. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*.
- Daniel M. Bikel. 2004. A distributional analysis of a lexicalized statistical parsing model. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing (EMNLP 2004)*.
- Jari Björne, Juho Heimonen, Filip Ginter, Antti Airola, Tapio Pahikkala, and Tapio Salakoski. 2009. Extracting complex biological events with rich graph-based feature sets. In *Proceedings of the Association for Computational Linguistics Workshop on BioNLP: Shared Task*.
- Catherine Blake. 2007. The role of sentence structure in recognizing textual entailment. In *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, pages 101–106, Prague, June.
- Eugene Charniak and Mark Johnson. 2005. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the 43rd Annual Meeting of the ACL*.
- Eugene Charniak. 2000. A maximum-entropy-inspired parser. In *Proceedings of NAACL-2000*.
- Andrew B. Clegg and Adrian J. Shepherd. 2007. Benchmarking natural-language parsers for biological applications using dependency graphs. *BMC Bioinformatics*, 8(24).
- Marie-Catherine de Marneffe and Christopher D. Manning. 2008. The Stanford typed dependencies representation. In *COLING Workshop on Cross-framework and Cross-domain Parser Evaluation*.
- Marie-Catherine de Marneffe, Bill MacCartney, and Christopher D. Manning. 2006. Generating typed dependency parses from phrase structure parses. In *5th International Conference on Language Resources and Evaluation (LREC 2006)*.
- Jason S. Kessler. 2008. Polling the blogosphere: a rule-based approach to belief classification. In *In-*

- ternational Conference on Weblogs and Social Media.*
- Dan Klein and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of the 41st Meeting of the Association for Computational Linguistics*.
- Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajic. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of HLT-EMNLP*.
- Yusuke Miyao, Rune Saetre, Kenji Sagae, Takuya Matsuzaki, and Jun'ichi Tsujii. 2008. Task-oriented evaluation of syntactic parsers and their representations. In *Proceedings of ACL-08:HLT*.
- Joakim Nivre, Johan Hall, and Jens Nilsson. 2006. MaltParser: A data-driven parser-generator for dependency parsing. In *Proceedings of the fifth international conference on Language Resources and Evaluation (LREC 2006)*.
- Sampo Pyysalo, Filip Ginter, Katri Haverinen, Juho Heimonen, Tapio Salakoski, and Veronika Laippala. 2007. On the unification of syntactic annotations under the Stanford dependency scheme: A case study on BioInfer and GENIA. In *Proceedings of BioNLP 2007: Biological, translational, and clinical language processing (ACL07)*.
- Mike Ross and Linas Vepstas, 2008. *RelEx*. OpenCog Project. <http://opencog.org/wiki/RelEx>.
- Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of HLT-NAACL 2003*.
- Sofie Van Landeghem, Yvan Saeys, Bernard De Baets, and Yves Van de Peer. 2009. Analyzing text in search of bio-molecular events: a high-precision machine learning framework. In *Proceedings of the Association for Computational Linguistics Workshop on BioNLP: Shared Task*.