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

Daniel Cer[⋆], Marie-Catherine de Marneffe[‡], Daniel Jurafsky[‡], Christopher D. Manning[⋆‡]

[⋆]Computer Science Department and [‡]Linguistics Department
Stanford University
Stanford, CA 94305, USA
{cerd,mcdm,jurafsky,manning}@stanford.edu

Abstract

We investigate a number of approaches to generating Stanford Dependencies, a widely used semantically-oriented dependency representation. We examine algorithms specifically designed for dependency parsing (Nivre, Nivre Eager, Covington, Eisner, and RelEx) as well as dependencies extracted from constituent parse trees created by phrase structure parsers (Charniak, Charniak-Johnson, Bikel, Berkeley and Stanford). We found that phrase structure parsers systematically outperform algorithms designed specifically for dependency parsing. The most accurate method for generating dependencies is the Charniak-Johnson reranking parser, with 89% (labeled) attachment F1 score. The fastest methods are Nivre, Nivre Eager, and Covington. When used with a linear classifier to make local parsing decisions, these methods can parse the entire Penn Treebank development set (section 22) in less than 10 seconds on an Intel Xeon E5520. However, this speed comes with a substantial drop in F1 score (about 76% for labeled attachment) compared to competing methods. By tuning how much of the search space is explored by the Charniak-Johnson parser, we are able to arrive at a balanced configuration that is both fast and nearly as good as the most accurate approaches.

1. Introduction

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., Adams et al. (2007), Blake (2007), Banko et al. (2007), Harmeling (2007), Meena and Prabhakar (2007), Zouaq et al. (2007), Kessler (2008)) and the biomedical text mining community (i.a., Pyysalo et al. (2007), Greenwood and Stevenson (2007), Urbain et al. (2007), Giles and Wren (2008), Björne et al. (2009), Van Landeghem et al. (2009)). When the Stanford Dependencies are used as part of an applied system or when they must be constructed for a large quantity of text, it is often important not just that the dependency representation is accurate but also that it can be produced reasonably quickly.

Stanford Dependencies have traditionally been extracted from constituent parses. 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. It is reasonable to expect that approaches specifically designed for dependency parsing, such as Eisner (Eisner, 1996), Covington (Covington, 2001), minimum spanning tree (MST) (McDonald et al., 2005), and Nivre (Nivre, 2003), would be faster, given that these approaches have lower algorithmic time complexity.[1] However, it is uncertain how much faster these algorithms perform in practice and how their speed and accuracy compare both to each other and to the standard approach of using a constituent parser.

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; Clegg, 2008). Miyao et al. (2008) developed the approach of automatically converting parsers' default output into dependency representations to evaluate 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. 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 an analysis of characteristic errors of each.

2. Methods

Experiments are performed on the Penn Treebank using a dual CPU Intel Xeon E5520. Parsers are trained using the standard training set of the Penn Treebank consisting of sections 2 through 21. We compare five popular state-of-the-art constituent parsers: Stanford englishPCFG v1.6.2 (Klein and Manning, 2003), Charniak 05Aug16 (Charniak, 2000), Charniak-Johnson June06 (CJ) (Charniak and Johnson, 2005), Bikel v1.2 (Bikel, 2004) and Berkeley v1.1 (Petrov et al., 2006). 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), the implementation of the Eisner algoritm provided by the MSTParser 0.4.3b (Eisner, 1996; 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 using the default feature set for each algorithm. The basic dependencies provide projective grammatical relations between every word in the sentence,

[1]Given a sentence of length n, the time required by a lexicalized parser implemented using CKY will scale on the order of $O(n^5)$. In the case of dependency parsing, the time complexities are $O(n^3)$ for Eisner, $O(n^2)$ for Covington, and $O(n)$ for Nivre.

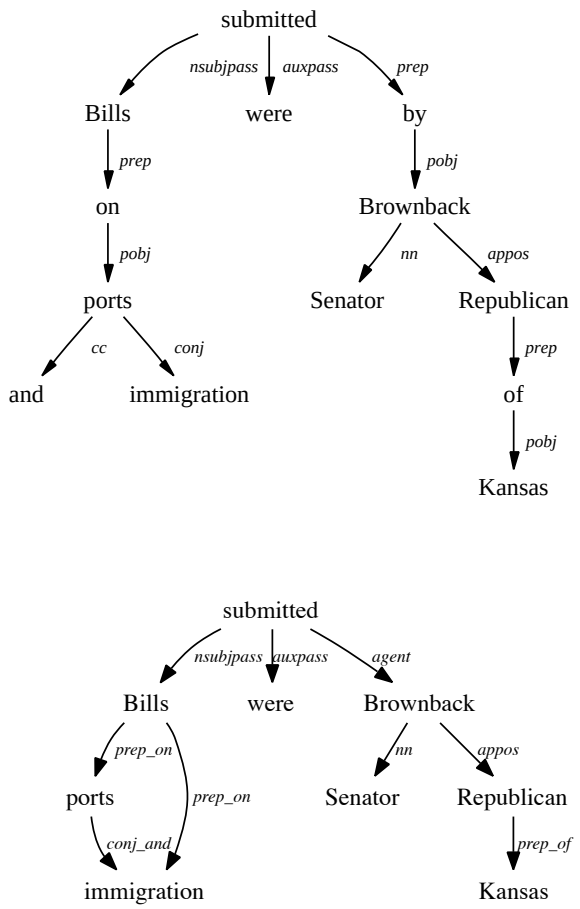

Figure 1: Basic and standard Stanford dependency representations for the sentence *Bills on ports and immigration were submitted by Senator Brownback, Republican of Kansas.*

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 which features collapsing of dependencies involving prepositions and conjunctions, as well as propagation of dependencies between conjuncts. Figure 1 shows the two dependency representations, basic and standard, for the sentence *Bills on ports and immigration were submitted by Senator Brownback, Republican of Kansas.*

Directly training non-projective parsing models, such as MST algorithm 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.

3. Results

Table 1 reports attachment F1 score for the different parsers on section 22 of the Penn TreeBank using the standard Stanford dependency representation (i.e., with collapsing and propagation of dependencies). Table 2 reports the corresponding attachment precision and recall scores. We use F1 score rather than attachment accuracy since the standard

Threads	Parse time
1	10:18
2	5:45
4	15:20

Table 4: Multithreading performance of the CJ reranking parser using the default search space size (T210). While running with 2 threads improves the speed of the parser, using more actually slows the parser down.

Stanford dependency representation allows each word to have multiple governors and parsers may generate a different number of dependencies for each sentence. "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 v2.0 with the MEMM tagging model (left3words-wsj-0-18) (Toutanova et al., 2003). To better take advantage of multicore machines, the CJ parser defaults to using 2 threads. However, to make the comparison fair with the other parsers, only one thread was used here. Multithreading results are presented below.

The dependencies extracted from the constituent parsers are the most accurate, but they are also the slowest to generate. The best performing parser is CJ reranking. However, it is followed closely by both Berkeley and Charniak. The performance of CJ reranking and Charniak is not surprising given that these parsers have been adapted over the years to do well parsing the English Penn Treebank. Interestingly, Berkeley, which is a newer and more general parser, is competitive in performance as well as in speed.

The fastest parsers are those included in the Malt package, Nivre, Nivre Eager, and Covington, when interactions between model features are not used.[2] Nivre Eager with feature interactions and MSTParser (Eisner) achieve better F1 scores than the other dependency parsers, and come closer to the scores obtained by constituent parsers. They are also much faster than the constituent parsers. Nivre Eager with feature interactions is about 67% faster than Berkeley, the fastest constituent parser. MSTParser (Eisner) is around 40% faster than Berkeley.

Both the Charniak and the CJ parsers allow users to trade off parsing accuracy for speed by adjusting how liberal

[2] As released, the MaltParser (v1.3) has a bug that causes parse time with liblinear models to be quadratic in *the number of words in the corpus being parsed* due to pre-insertion in an array list that grows with each parsing prediction made. The results presented here are from our own patched version, which is about 2 orders of magnitude faster than the v1.3 release on the data sets reported here. This bug is fixed in the v1.3.1 release. The speed of the MaltParser is 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 this fix has been applied.

Type	Parser	Attachment F1		Time			
		Unlabeled	Labeled	POS tag	Parse	Dep. extraction	Total
Constituent	Stanford	87.2	84.2	–	10:04	1:01	11:05
	Charniak	90.5	87.8	–	11:09	1:01	12:10
	CJ	**91.7**	**89.1**	–	10:18	1:00	11:18
	Bikel	88.7	85.3	–	28:57	1:00	29:57
	Berkeley	90.5	87.9	–	9:14	1:00	10:14
Dependency	Covington	80.0	76.6	0:03	**0:09**	0:04	**0:16**
	Nivre Eager	80.1	76.2	0:03	**0:08**	0:05	**0:16**
	Nivre	80.2	76.3	0:03	**0:08**	0:04	**0:15**
	Nivre Eager Feature Interact	**84.8**	**81.1**	0:03	3:15	0:05	3:23
	MSTParser (Eisner)	82.6	78.8	0:03	5:54	0:04	6:01
	RelEx	57.8	48.1	–	31:38	–	31:38

Table 1: Unlabeled and labeled attachment F1 score (%) and time (min:seconds) to generate standard 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.

Type	Parser	Unlabeled attachment		Labeled attachment	
		P	R	P	R
Constituent	Stanford	87.3	87.1	84.2	84.1
	Charniak	90.5	90.4	87.8	87.7
	CJ	**91.7**	**91.7**	**89.2**	**89.1**
	Bikel	88.9	88.6	85.4	85.1
	Berkeley	90.6	90.5	88.0	87.9
Dependency	Covington	80.9	79.1	77.5	75.7
	Nivre Eager	80.6	79.5	76.8	75.7
	Nivre	80.7	79.8	76.8	75.9
	Nivre Eager Feature Interact	**85.4**	**84.2**	**81.7**	**80.5**
	MSTParser (Eisner)	83.0	82.2	79.2	78.4
	RelEx	70.4	49.1	58.6	40.8

Table 2: Unlabeled and labeled attachment precision and recall (%) to generate standard Stanford Dependencies with different types of parsers (constituent vs. dependency).

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 3, by adjusting this parameter down, the time required can be reduced to nearly that of the fastest dependency parsing algorithms. Unfortunately, these gains come with a sizable reduction in dependency accuracy. However, by modestly 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. When multiple CPU cores are available, the speed of the CJ parser can also be improved by using multiple threads. Table 4 shows the parse time for the CJ parser when using 1 to 4 threads. Parsing speed nearly doubles when 2 threads are used instead of 1. However, increasing the threads to 4 results in much slower performance than just using 1 thread.[3]

[3]The dual CPU E5520 we used for our experiments has a total of 8 CPU cores. On this machine, a good threading implementation might show speed gains using up to 8 threads. It is worth noting that a near ideal 8x speedup can be obtained for all of the parsers presented here by simply starting multiple parsing jobs on the machine with each job being assigned to a different slice of the corpus to be parsed.

4. Error analysis

We performed error analysis on section 22 of the Penn TreeBank, the same data used for table 1. The very low score from RelEx is largely due to the parser omitting a sizable number of dependencies, as can be seen in the recall results in table 2. However, the dependencies it produces are still less accurate than those from other parsers.

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. The latter is produced when the dependency extraction code has difficulty labeling a relationship within a parse tree. 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.*, Berkeley, Stanford, Charniak and CJ misattach the adverbial clause: *advcl*(sold-1, sold-2) instead of *advcl*(maintain, sold-2). Berkeley, Stanford, Bikel and CJ produce *xcomp*(requires, maintain) instead of *infmod*(capital, maintain). In (2) *The*

Parser	Attachment F1		Time			
	Unlabeled	Labeled	POS tag	Parse	Dep. extraction	Total
Charniak T10	79.7	75.7	–	0:14	1:00	1:14
Charniak T50	89.5	86.7	–	2:06	1:03	3:09
CJ T10	80.1	76.1	–	1:18	0:59	2:17
CJ T50	90.4	87.6	–	2:31	1:01	3:32

Table 3: Unlabeled and labeled attachment F1 score (%) and time (min:seconds) to generate Stanford Dependencies with different beams of the Charniak and Charniak-Johnson parsers.

decline in the German Stock Index of 203.56 points, or 12.8%, to 1385.72 was the Frankfurt market's steepest fall ever., all the constituent parsers misattach *points* to *Index* with the relation *prep_of*. For *1385.82* however, CJ and Bikel do get the right phrase structure and correctly produce *prep_to*(decline, 1385.82).

Decreasing the beam size for the CJ parser to T10 leads to a greater number of such errors. Recall and precision for the following dependencies especially suffer: adverbial clauses (*advcl*), appositions (*appos*), indirect objects (*iobj*), clausal and nominal subjects (*csubj*, *csubjpass*, *nsubj*, *nsubjpass*), relative clauses (*rcmod*, *rel*), prepositional phrases as well as infinitival modifiers (*infmod*), participial modifiers (*partmod*) and quantifier modifiers (*quantmod*). However, when only decreasing the beam size to T50, there are no substantial differences in recall and precision for specific dependencies, except for the ones involving prepositional phrases: the prepositions are wrongly attached more often than when the default beam size (T210) is used. CJ achieves substantially better precision and recall than the other constituent parsers for infinitival modifiers (*infmod*) and relative clauses (*rcmod*). Berkeley performs better for the *parataxis* relation.

Nivre, Nivre Eager, and Covington often produce more local attachments than both the constituent parsers and MSTParser (Eisner). For example, in (3) *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) for Nivre, Nivre Eager and Covington, whereas MSTParser (Eisner) and Nivre Eager with feature interactions get it right. Incorrect higher attachments sometimes occur, probably due to a lexical preference: in example (1), Nivre Eager and Covington give *rcmod*(assets-1, sold-1) instead of *rcmod*(thrifts, sold-1). Nivre and MSTParser (Eisner) find the correct relation. 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, the Malt algorithms rarely give such output, presumably because locally the attachment to the copula appears to be reasonable.

When comparing recall and precision for specific dependencies between the Malt algorithms, the only noticeable difference is that Covington produces better numbers for infinitival modifiers (*infmod*), purpose clauses (*purpcl*) and relative (*rel*). CJ T50 attains even better accuracy for these relations, except for *infmod* for which it has better precision (84% vs. 63%) but slightly worse recall (69% vs. 73%).

Most systematic errors made by the dependency parsers included in the Malt package can be attributed to their deterministic nature: once they mistakenly attach a dependent that looks good given the local context and the partial intermediate parse, they cannot backtrack even if it forces the parser to make unusual subsequent attachments. Lexical preference can then conspire with locality and introduce parse errors.

Even though MSTParser (Eisner) 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 (Eisner) which favors short distance dependencies. As a result in sentence (2), all dependency parsers wrongly misattach *points* to the neighbor *Index* with the relation *prep_of* instead of attaching it higher to *decline*.

5. Conclusion

For the Stanford Dependencies, constituent parsers appear to systematically outperform algorithms designed specifically for dependency parsing. 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, Charniak-Johnson reranking, and Berkeley parsers is so high that, in the vast majority of cases, dependency parse consumers are better off using them, and then converting the output to typed 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.

Acknowledgements

This work was supported in part by the Air Force Research Laboratory (AFRL) under prime contract no. FA8750-09-C-0181 and by the Defense Advanced Research Projects Agency through IBM. The content does not necessarily reflect the views of the U.S. Government, and no official endorsement should be inferred.

6. References

Rod Adams, Gabriel Nicolae, Cristina Nicolae, and Sanda Harabagiu. 2007. Textual entailment through extended lexical overlap and lexico-semantic matching. In *Proceedings of the ACL-PASCAL Workshop on Textual En-*

tailment and Paraphrasing, pages 119–124, Prague, June.

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).

Andrew B. Clegg. 2008. *Computational-Linguistic Approaches to Biological Text Mining*. Ph.D. thesis, School of Crystallography, Birkbeck, University of London.

Michael A. Covington. 2001. A fundamental algorithm for dependency parsing. In *Proceedings of the 39th Annual ACM Southeast Conference*.

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 Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING-96)*, pages 340–345, Copenhagen, August.

Cory B. Giles and Jonathan D. Wren. 2008. Large-scale directional relationship extraction and resolution. *BMC Bioinformatics*, 9(Suppl 9):S11.

Mark A. Greenwood and Mark Stevenson. 2007. A semi-supervised approach to learning relevant protein-protein interaction articles. In *Proceedings of the Second BioCreAtIvE Challenge Workshop, Madrid, Spain*.

Stefan Harmeling. 2007. An extensible probabilistic transformation-based approach to the third recognizing textual entailment challenge. In *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, pages 137–142, Prague, June.

Jason S. Kessler. 2008. Polling the blogosphere: a rule-based approach to belief classification. In *International 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*.

Arun Meena and T. V. Prabhakar. 2007. Sentence level sentiment analysis in the presence of conjuncts using linguistic analysis. In *Advances in Information Retrieval*, volume 4425 of *Lecture Notes in Computer Science*. Springer.

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)*.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT 03)*.

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. 2006. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of COLING-ACL 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*.

Jay Urbain, Nazli Goharian, and Ophir Frieder. 2007. IIT TREC 2007 genomics track: Using concept-based semantics in context for genomics literature passage retrieval. In *The Sixteenth Text REtrieval Conference (TREC 2007) Proceedings*.

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*.

Amal Zouaq, Roger Nkambou, and Claude Frasson. 2007. Building domain ontologies from text for educational purposes. In *Proceedings of the Second European Conference on Technology Enhanced Learning: Creating new learning experiences on a global scale*.