It Depends: Dependency Parser Comparison Using A Web-based Evaluation Tool

Jinho D. Choi
Emory University
400 Dowman Dr.
Atlanta, GA 30322, USA
jchoi31@emory.edu

Joel Tetreault
Yahoo Labs
229 West 43rd St.
New York, NY 10036, USA
tetreaul@yahoo-inc.com

Amanda Stent
Yahoo Labs
229 West 43rd St.
New York, NY 10036, USA
stent@yahoo-inc.com

Abstract

The last few years have seen a surge in the number of accurate, fast, publicly available dependency parsers. At the same time, the use of dependency parsing in NLP applications has increased. It can be difficult for a non-expert to select a good "off-the-shelf" parser. We present a comparative analysis of ten leading statistical dependency parsers on a multi-genre corpus of English. For our analysis, we developed a new web-based tool that gives a convenient way of comparing dependency parser outputs. Our analysis will help practitioners choose a parser to optimize their desired speed/accuracy tradeoff, and our tool will help practitioners examine and compare parser output.

1 Introduction

Dependency parsing is a valuable form of syntactic processing for NLP applications due to its transparent lexicalized representation and robustness with respect to flexible word order languages. Thanks to over a decade of research on statistical dependency parsing, many dependency parsers are now publicly available. In this paper, we report on a comparative analysis of leading statistical dependency parsers using a multi-genre corpus. Our purpose is not to introduce a new parsing algorithm but to assess the performance of existing systems across different genres of language use and to provide tools and recommendations that practitioners can use to choose a dependency parser. The contributions of this work include:

- A comparison of the accuracy and speed of ten state-of-the-art dependency parsers, covering a range of approaches, on a large multi-genre corpus of English.
- A new web-based tool, DEPENDABLE, for side-by-side comparison and visualization of the output from multiple dependency parsers.
- A detailed error analysis for these parsers using DEPENDABLE, with recommendations for parser choice for different factors.
- The release of the set of dependencies used in our experiments, the test outputs from all parsers, and the parser-specific models.

2 Related Work

There have been several shared tasks on dependency parsing conducted by CoNLL (Buchholz and Marsi, 2006; Nivre and others, 2007; Surdeanu and others, 2008; Hajič and others, 2009), SANCL (Petrov and McDonald, 2012), SPMRL (Seddah and others, 2013), and SemEval (Oepen and others, 2014). These shared tasks have led to the public release of numerous statistical parsers. The primary metrics reported in these shared tasks are: labeled attachment score (**LAS**) – the percentage of predicted dependencies where the arc and the label are assigned correctly; unlabeled attachment score (**UAS**) – where the arc is assigned correctly; label accuracy score (**LS**) – where the label is assigned correctly; and exact match (**EM**) – the percentage of sentences whose predicted trees are entirely correct.

Although shared tasks have been tremendously useful for advancing the state of the art in dependency parsing, most English evaluation has employed a single-genre corpus, the WSJ portion of the Penn Treebank (Marcus et al., 1993), so it is not immediately clear how these results gen-

Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing, pages 387–396, Beijing, China, July 26-31, 2015.

	BC	BN	MZ	NW	PT	TC	WB	ALL
Training	171,120	206,057	163,627	876,399	296,437	85,466	284,975	2,084,081
Development	29,962	25,274	15,422	147,958	25,206	11,467	36,351	291,640
Test	35,952	26,424	17,875	60,757	25,883	10,976	38,490	216,357
Training	10,826	10,349	6,672	34,492	21,419	8,969	12,452	105,179
Development	2,117	1,295	642	5,896	1,780	1,634	1,797	15,161
Test	2,211	1,357	780	2,327	1,869	1,366	1,787	11,697

Table 1: Distribution of data used for our experiments. The first three/last three rows show the number of tokens/trees in each genre. BC: broadcasting conversation, BN: broadcasting news, MZ: news magazine, NW: newswire, PT: pivot text, TC: telephone conversation, WB: web text, ALL: all genres combined.

eralize.[1] Furthermore, a detailed comparative error analysis is typically lacking. The most detailed comparison of dependency parsers to date was performed by McDonald and Nivre (2007; 2011); they analyzed accuracy as a function of sentence length, dependency distance, valency, non-projectivity, part-of-speech tags and dependency labels.[2] Since then, additional analyses of dependency parsers have been performed, but either with respect to specific linguistic phenomena (e.g. (Nivre et al., 2010; Bender et al., 2011)) or to downstream tasks (e.g. (Miwa and others, 2010; Petrov et al., 2010; Yuret et al., 2013)).

3 Data

3.1 OntoNotes 5

We used the English portion of the OntoNotes 5 corpus, a large multi-lingual, multi-genre corpus annotated with syntactic structure, predicate-argument structure, word senses, named entities, and coreference (Weischedel and others, 2011; Pradhan and others, 2013). We chose this corpus rather than the Penn Treebank used in most previous work because it is larger (2.9M vs. 1M tokens) and more diverse (7 vs. 1 genres). We used the standard data split used in CoNLL'12 [3], but removed sentences containing only one token so as not to artificially inflate accuracy.

Table 1 shows the distribution across genres of training, development, and test data. For the most strict and realistic comparison, we trained all ten parsers using automatically assigned POS tags from the tagger in ClearNLP (Choi and Palmer, 2012a), which achieved accuracies of 97.34 and 97.52 on the development and test data, respectively. We also excluded any "morphological" feature from the input, as these are often not available in non-annotated data.

3.2 Dependency Conversion

OntoNotes provides annotation of constituency trees only. Several programs are available for converting constituency trees into dependency trees. Table 2 shows a comparison between three of the most widely used: the LTH (Johansson and Nugues, 2007),[4], Stanford (de Marneffe and Manning, 2008),[5] and ClearNLP (Choi and Palmer, 2012b)[6] dependency converters. Compared to the Stanford converter, the ClearNLP converter produces a similar set of dependency labels but generates fewer unclassified dependencies (0.23% vs. 3.62%), which makes the training data less noisy.

Both the LTH and ClearNLP converters produce long-distance dependencies and use function tags for the generation of dependency relations, which allows one to generate rich dependency structures including non-projective dependencies. However, only the ClearNLP converter adapted the new Treebank guidelines used in OntoNotes. It can also produce secondary dependencies (e.g. right-node raising, referent), which can be used for further analysis. We used the ClearNLP converter to produce dependencies for our experiments.

	LTH	Stanford	ClearNLP
Long-distance	✓		✓
Secondary	1	2	4
Function tags	✓		✓
New TB format			✓

Table 2: Dependency converters. The "secondary" row shows how many types of secondary dependencies that can be produced by each converter.

[1] The SANCL shared task used OntoNotes and the Web Treebanks instead for better generalization.

[2] A detailed error analysis of constituency parsing was performed by (Kummerfeld and others, 2012).

[3] conll.cemantix.org/2012/download/ids/

[4] http://nlp.cs.lth.se/software

[5] http://nlp.stanford.edu/software

[6] http://www.clearnlp.com

Parser	Approach	Language	License
ClearNLP v2,3[7]	Transition-based, selectional branching (Choi and McCallum, 2013)	Java	Apache
GN13[8]	Easy-first, dynamic oracle (Goldberg and Nivre, 2013)	Python	GPL v2
LTDP v2.0.3[9]	Transition-based, beam-search + dynamic prog. (Huang et al., 2012)	Python	n/a
Mate v3.6.1[10]	Maximum spanning tree, 3rd-order features (Bohnet, 2010)	Java	GPL v2
RBG[11]	Tensor decomposition, randomized hill-climb (Lei et al., 2014)	Java	MIT
Redshift[12]	Transition-based, non-monotonic (Honnibal et al., 2013)	Cython	FOSS
spaCy[13]	Transition-based, greedy, dynamic oracle, Brown clusters	Cython	Dual
SNN[14]	Transition-based, word embeddings (Chen and Manning, 2014)	Java	GPL v2
Turbo v2.2[15]	Dual decomposition, 3rd-order features (Martins et al., 2013)	C++	GPL v2
Yara[16]	Transition-based, beam-search, dynamic oracle (Rasooli and Tetreault, 2015)	Java	Apache

Table 3: Dependency parsers used in our experiments.

4 Parsers

We compared ten state of the art parsers representing a wide range of contemporary approaches to statistical dependency parsing (Table 3). We trained each parser using the training data from OntoNotes. For all parsers we trained using the automatic POS tags generated during data preprocessing, as described above.

Training settings For most parsers, we used the default settings for training. For the SNN parser, following the recommendation of the developers, we used the word embeddings from (Collobert and others, 2011).

Development data ClearNLP, LTDP, SNN and Yara make use of the development data (for parameter tuning). Mate and Turbo self-tune parameter settings using the training data. The others were trained using their default/"standard" parameter settings.

Beam search ClearNLP, LTDP, Redshift and Yara have the option of different beam settings. The higher the beam size, the more accurate the parser usually becomes, but typically at the expense of speed. For LTDP and Redshift, we experimented with beams of 1, 8, 16 and 64 and found that the highest accuracy was achieved at beam 8.[17] For ClearNLP and Yara, a beam size of 64 produced the best accuracy, while a beam size of 1 for LTDT, ClearNLP, and Yara produced the best speed performance. Given this trend, we also include how those three parsers perform at beam 1 in our analyses.

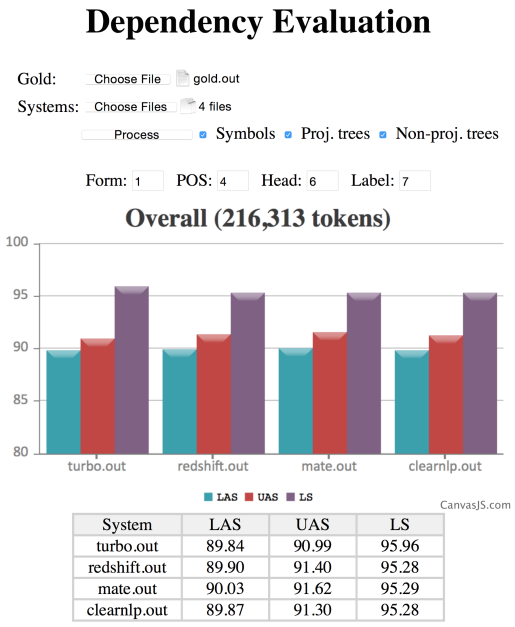
Feature Sets RBG, Turbo and Yara have the options of different feature sets. A more complex or larger feature set has the advantage of accuracy, but often at the expense of speed. For RBG and Turbo, we use the "Standard" setting and for Yara, we use the default ("not basic") feature setting.

Output All the parsers other than LTDP output labeled dependencies. The ClearNLP, Mate, RBG, and Turbo parsers can generate non-projective dependencies.

5 DEPENDABLE: Web-based Evaluation and Visualization Tool

There are several very useful tools for evaluating the output of dependency parsers, including the venerable `eval.pl`[18] script used in the CoNLL shared tasks, and newer Java-based tools that support visualization of and search over parse trees such as TedEval (Tsarfaty et al., 2011),[19] MaltEval (Nilsson and Nivre, 2008)[20] and "What's wrong with my NLP?".[21] Recently, there is momentum towards web-based tools for annotation and visualization of NLP pipelines (Stenetorp and others, 2012). For this work, we used a new web-based tool, DEPENDABLE, developed by the first author of this paper. It requires no installation and so provides a convenient way to evaluate and compare dependency parsers. The following are key features of DEPENDABLE:

[7] www.clearnlp.com
[8] cs.bgu.ac.il/~yoavg/software/sdparser
[9] acl.cs.qc.edu/~lhuang
[10] code.google.com/p/mate-tools
[11] github.com/taolei87/RBGParser
[12] github.com/syllog1sm/Redshift
[13] honnibal.github.io/spaCy
[14] nlp.stanford.edu/software/nndep.shtml
[15] www.ark.cs.cmu.edu/TurboParser
[16] https://github.com/yahoo/YaraParser
[17] Due to memory limitations we were unable to train Redshift on a beam size greater than 8.
[18] ilk.uvt.nl/conll/software.html
[19] www.tsarfaty.com/unipar/
[20] www.maltparser.org/malteval.html
[21] whatswrong.googlecode.com



System	LAS	UAS	LS
turbo.out	89.84	90.99	95.96
redshift.out	89.90	91.40	95.28
mate.out	90.03	91.62	95.29
clearnlp.out	89.87	91.30	95.28

Figure 1: Screenshot of our evaluation tool.

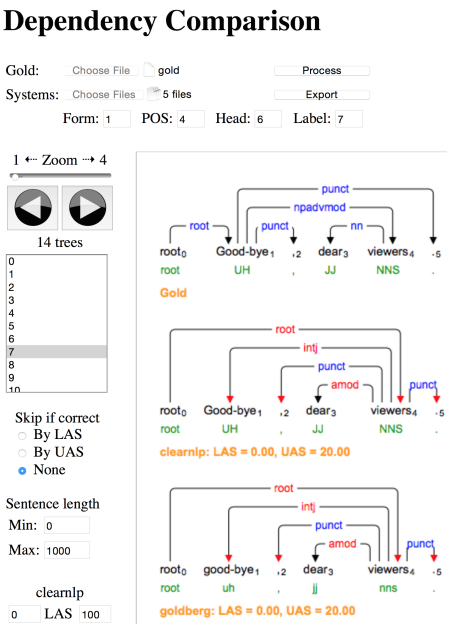


Figure 2: Screenshot of our visualization tool.

- It reads any type of Tab Separated Value (TSV) format, including the CoNLL formats.
- It computes LAS, UAS and LS for parse outputs from multiple parsers against gold (manual) parses.
- It computes exact match scores for multiple parsers, and "oracle ensemble" output, the upper bound performance obtainable by combining all parser outputs.
- It allows the user to exclude symbol tokens, projective trees, or non-projective trees.
- It produces detailed analyses by POS tags, dependency labels, sentence lengths, and dependency distances.
- It reports statistical significance values for all parse outputs (using McNemar's test).

DEPENDABLE can be also used for visualizing and comparing multiple dependency trees together (Figure 2). A key feature is that the user may select parse trees by specifying a range of accuracy scores; this enabled us to perform the error analyses in Section 6.5. DEPENDABLE allows one to filter trees by sentence length and highlights arc and label errors. The evaluation and comparison tools are publicly available at `http://nlp.mathcs.emory.edu/clearnlp/dependable`.

6 Results and Error Analysis

In this section, we report overall parser accuracy and speed. We analyze parser accuracy by sentence length, dependency distance, non-projectivity, POS tags and dependency labels, and genre. We report detailed manual error analyses focusing on sentences that multiple parsers parsed incorrectly.[22] All analyses, other than parsing speed, were conducted using the DEPENDABLE tool.[23] The full set of outputs from all parsers, as well as the trained models for each parser, available at `http://amandastent.com/dependable/`.

We also include the greedy parsing results of ClearNLP, LTDP, and Yara in two of our analyses to better illustrate the differences between the greedy and non-greedy settings. The greedy parsing results are denoted by the subscript '$_g$'. These two analyses are the overall accuracy results, presented in Section 6.1 (Table 4), and the overall speed results, presented in Section 6.2 ((Table 5 and Figure). All other analyses exclude the ClearNLP$_g$, LTDP$_g$ and Yara$_g$.

[22] For one sentence in the NW data, the LTDP parser failed to produce a complete parse containing all tokens, so we removed this sentence for all parsers, leaving 11,696 trees (216,313 tokens) in the test data.

[23] We compared the results produced by DEPENDABLE with those produced by `eval07.pl`, and verified that LAS, UAS, LA, and EM were the same when punctuation was included. Our tool uses a slightly different symbol set than `eval07.pl`: `!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~`

	With Punctuation						Without Punctuation					
	Overall			Exact Match			Overall			Exact Match		
	LAS	UAS	LS	LAS	UAS	LS	LAS	UAS	LS	LAS	UAS	LS
ClearNLP$_g$	**89.19**	**90.63**	**94.94**	**47.65**	**53.00**	**61.17**	**90.09**	**91.72**	**94.29**	**49.12**	**55.01**	**61.31**
GN13	87.59	89.17	93.99	43.78	48.89	56.71	88.75	90.54	93.32	45.44	51.20	56.88
LTDP$_g$	n/a	85.75	n/a	n/a	46.38	n/a	n/a	87.16	n/a	n/a	48.01	n/a
SNN	86.42	88.15	93.54	42.98	48.53	55.87	87.63	89.59	92.70	43.96	49.83	55.91
spaCy	87.92	89.61	94.08	43.36	48.79	55.67	88.95	90.86	93.32	44.97	51.28	55.70
Yara$_g$	85.93	87.64	92.99	42.94	47.77	54.79	87.39	89.32	92.24	44.25	49.44	54.96
ClearNLP	89.87	91.30	95.28	49.38	55.18	63.18	90.64	92.26	**94.67**	50.61	56.88	63.24
LTDP	n/a	88.18	n/a	n/a	51.62	n/a	n/a	89.17	n/a	n/a	53.54	n/a
Mate	**90.03**	**91.62**	**95.29**	49.66	**56.44**	62.71	**90.70**	**92.50**	**94.67**	50.83	**58.36**	62.72
RBG	89.57	91.45	94.71	46.49	55.49	58.45	90.23	92.35	94.01	47.64	56.54	58.07
Redshift	89.48	91.01	95.04	49.71	55.82	62.70	90.27	92.00	94.42	50.88	57.28	62.78
Turbo	89.81	91.50	95.00	48.08	55.33	60.49	90.49	92.40	94.34	49.29	57.09	60.52
Yara	89.80	91.36	95.19	**50.07**	56.18	**63.36**	90.47	92.24	94.57	**51.02**	57.53	**63.42**

Table 4: Overall parsing accuracy. The top 6 rows and the bottom 7 rows show accuracies for greedy and non-greedy parsers, respectively.

6.1 Overall Accuracy

In Table 4, we report overall accuracy for each parser. For clarity, we report results separately for greedy and non-greedy versions of the parsers. Over all the different metrics, MATE is a clear winner, though ClearNLP, RBG, Redshift, Turbo and Yara are very close in performance. Looking at only the greedy parsers, ClearNLP$_g$ shows a significant advantage over the others.

We conducted a statistical significance test for the the parsers (greedy versions excluded). All LAS differences are statistically significant at $p < .01$ (using McNemar's test), except for: RBG vs. Redshift, Turbo vs. Yara, Turbo vs. ClearNLP and Yara vs. ClearNLP. All UAS differences are statistically significant at $p < .01$ (using McNemar's test), except for: SNN vs. LTDP, Turbo vs. Redshift, Yara vs. RBG and ClearNLP vs. Yara.

6.2 Overall Speed

We ran timing experiments on a 64 core machine with 16 Intel Xeon E5620 2.40 GHz processors and 24G RAM, and used the unix `time` command to time each run. Some parsers are multi-threaded; for these, we ran in single-thread mode (since any parser can be externally parallelized). Most parsers do not report model load time, so we first ran each parser five times with a test set of 10 sentences, and then averaged the middle three times to get the average model load time.[24] Next, we ran each parser five times with the entire test set and derived the overall parse time by averaging the middle three parse times. We then subtracted the average model time from the average parse time and averaged over the number of sentences and tokens.

	Sent/Sec	Tokens/Sec	Language
ClearNLP$_g$	555	10,271	Java
GN13	95	1,757	Python
LTDP$_g$	232	4,287	Python
SNN	465	8,602	Java
spaCy	**755**	**13,963**	Cython
Yara$_g$	532	9,838	Java
ClearNLP	72	1,324	Java
LTDP	26	488	Python
Mate	30	550	Java
RBG	57	1,056	Java
Redshift	**188**	**3,470**	Cython
Turbo	19	349	C++
Yara	18	340	Java

Table 5: Overall parsing speed.

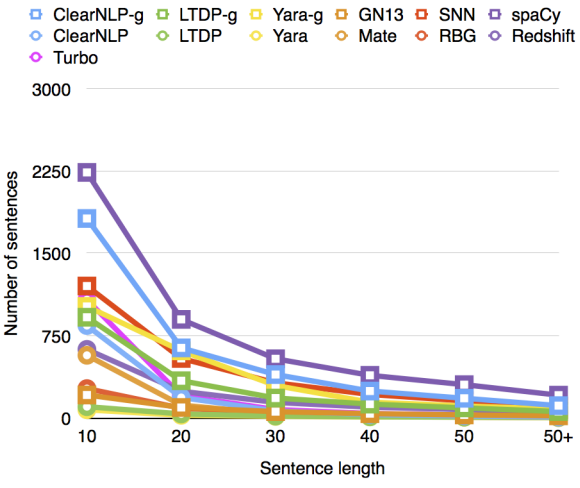


Figure 3: Number of sentences parsed per second by each parser with respect to sentence length.

Table 5 shows overall parsing speed for each parser. spaCy is the fastest greedy parser and Redshift is the fastest non-greedy parser. Figure 3

[24] Recall we exclude single-token sentences from our tests.

shows an analysis of parsing speed by sentence length in bins of length 10. As expected, as sentence length increases, parsing speed decreases remarkably.

6.3 Detailed Accuracy Analyses

For the following more detailed analyses, we used all tokens (including punctuation). As mentioned earlier, we exclude ClearNLP$_g$, LTDP$_g$ and Yara$_g$ from these analyses and instead use their respective non-greedy modes yielding higher accuracy.

Sentence Length We analyzed parser accuracy by sentence length in bins of length 10 (Figure 4). As expected, all parsers perform better on shorter sentences. For sentences under length 10, UAS ranges from 93.49 to 95.5; however, UAS declines to a range of 81.66 and 86.61 for sentence lengths greater than 50. The most accurate parsers (ClearNLP, Mate, RBG, Redshift, Turbo, and Yara) separate from the remaining when sentence length is more than 20 tokens.

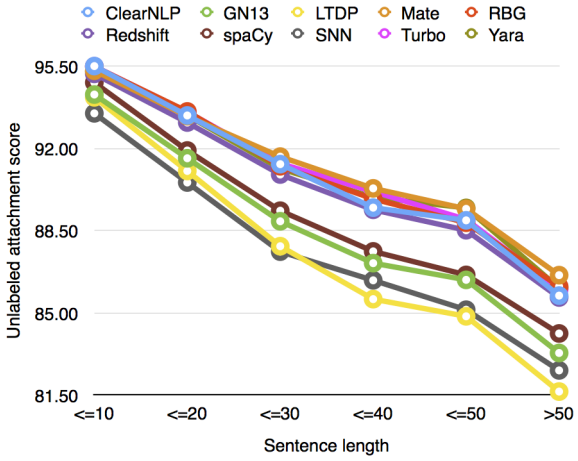


Figure 4: UAS by sentence length.

Dependency Distance We analyzed parser accuracy by dependency distance (depth from each dependent to its head; Figure 5). Accuracy falls off more slowly as dependency distance increases for the top 6 parsers vs. the rest.

Projectivity Some of our parsers only produce projective parses. Table 6 shows parsing accuracy for trees containing only projective arcs (11,231 trees, 202,521 tokens) and for trees containing non-projective arcs (465 trees, 13,792 tokens). As before, all differences are statistically significant at $p < .01$ except for: Redshift vs. RBG for overall LAS; LTDP vs. SNN for overall UAS; and Turbo vs. SpaCy for overall UAS. For strictly projective trees, the LTDP parser is 5th from the top in UAS. Apart from this, the grouping between "very good" and "good" parsers does not change.

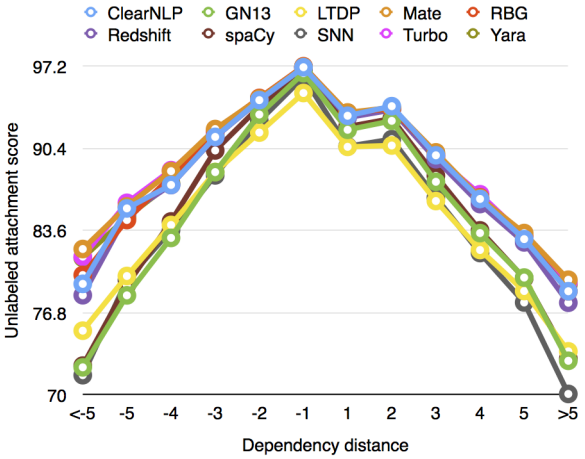


Figure 5: UAS by dependency distance.

	Projective only		Non-proj. only	
	LAS	UAS	LAS	UAS
ClearNLP	90.20	91.62	85.10	86.72
GN13	88.00	89.57	81.56	83.37
LTDP	n/a	90.24	n/a	57.83
Mate	**90.34**	**91.91**	**85.51**	**87.40**
RBG	89.86	91.72	84.83	86.94
Redshift	89.90	91.41	83.30	85.12
SNN	86.83	88.55	80.37	82.32
spaCy	88.31	89.99	82.15	84.08
Turbo	88.36	89.90	83.50	85.30
Yara	90.20	91.74	83.92	85.74

Table 6: Accuracy for proj. and non-proj. trees.

Dependency Relations We were interested in which dependency relations were computed with high/low overall accuracy, and for which accuracy varied between parsers. The dependency relations with the highest average LAS scores ($> 97\%$) were `possessive`, `hyph`, `expl`, `hmod`, `aux`, `det` and `poss`. These relations have strong lexical clues (e.g. `possessive`) or occur very often (e.g. `det`). Those with the lowest LAS scores ($< 50\%$) were `csubjpass`, `meta`, `dep`, `nmod` and `parataxis`. These either occur rarely or are very general (`dep`).

The most "confusing" dependency relations (those with the biggest range of accuracies across parsers) were `csubj`, `preconj`, `csubjpass`, `parataxis`, `meta` and `oprd` (all with a spread of $> 20\%$). The Mate and Yara parsers each had the highest accuracy for 3 out of the top 10 "confusing" dependency relations. The RBG parser

had the highest accuracy for 4 out of the top 10 "most accurate" dependency relations. SNN had the lowest accuracy for 5 out of the top 10 "least accurate" dependency relations, while the RBG had the lowest accuracy for another 4.

POS Tags We also examined error types by part of speech tag of the dependent. The POS tags with the highest average LAS scores ($> 97\%$) were the highly unambiguous tags POS, WP$, MD, TO, HYPH, EX, PRP and PRP$. With the exception of WP$, these tags occur frequently. Those with the lowest average LAS scores ($< 75\%$) were punctuation markers (,) and :, and the rare tags AFX, FW, NFP and LS.

Genres Table 7 shows parsing accuracy for each parser for each of the seven genres comprising the English portion of OntoNotes 5. Mate and ClearNLP are responsible for the highest accuracy for some genres, although accuracy differences among the top four parsers are generally small. Accuracy is highest for PT (pivot text, the Bible) and lowest for TC (telephone conversation) and WB (web data). The web data is itself multi-genre and includes translations from Arabic and Chinese, while telephone conversation data includes disfluencies and informal language.

6.4 Oracle Ensemble Performance

One popular method for achieving higher accuracy on a classification task is to use system combination (Björkelund and others, 2014; Le Roux and others, 2012; Le Roux et al., 2013; Sagae and Lavie, 2006; Sagae and Tsujii, 2010; Haffari et al., 2011). DEPENDABLE reports ensemble upper bound performance assuming that the best *tree* can be identified by an oracle (**macro**), or that the best *arc* can be identified by an oracle (**micro**). Table 8 provides an upper bound on ensemble performance for future work.

	LAS	**UAS**	**LS**
Macro	94.66	96.00	97.82
Micro	96.52	97.61	98.40

Table 8: Oracle ensemble performance.

The highest match was achieved between the RBG and Mate parser (62.22 UAS). ClearNLP, GN13 and LTDP all matched with Redshift the best, and RBG, Redshift and Turbo matched with Mate the best. SNN, spaCy and Turbo did not match well with other parsers; their respective "best match" score was never higher than 55.

6.5 Error Analysis

From the test data, we pulled out parses where only one parser achieved very high accuracy, and parses where only one parser had low accuracy (Table 9). As with the detailed performance analyses, we used the most accurate version of each parser for this analysis. Mate has the highest number of "generally good" parses, while the SNN parser has the highest number of "uniquely bad" parses. The SNN parser tended to choose the wrong root, but this did not appear to be tied to the number of verbs in the sentence - rather, the SNN parser just makes the earliest "reasonable" choice of root.

Parser UAS All others UAS	$\geq$ **90** $<$ **90**	$=$ **100** $<$ **90**	$<$ **90** $\geq$ **90**	$<$ **90** $=$ **100**
ClearNLP	42	11	45	15
LTDP	29	12	182	36
GN13	26	8	148	65
Mate	**75**	19	44	10
RBG	49	21	49	15
Redshift	38	17	28	8
SNN	70	**23**	**417**	**142**
spaCy	48	17	218	73
Turbo	54	15	28	14
Yara	33	15	27	7

Table 9: Differential parsing accuracies.

To further analyze these results, we first looked at the parse trees for "errorful" sentences where the parsers agreed. From the test data, we extracted parses for sentences where at least two parsers got UAS of $< 50\%$. This gave us 253 sentences. The distribution of these errors across genres varied: PT - 2.8%, MZ - 3.5%, BN - 9.8%, NW - 10.3%, WB - 17.4%, BC - 25.3%, TC - 30.8%.

By manual comparison using the DEPENDABLE tool, we identified frequently occurring potential sources of error. We then manually annotated all sentences for these error types. Figure 6 shows the number of "errorful" sentences of each type. Punctuation attachment "errors" are prevalent. For genres with "noisy" text (e.g. broadcast conversation, telephone conversation) a significant proportion of errors come from fragmented sentences or those containing backchannels or disfluencies. There are also a number of sentences with what appeared to be manual dependency labeling errors in the gold annotation.

	BC		BN		MZ		NW		PT		TC		WB	
	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS	LAS	UAS
ClearNLP	88.95	90.36	**89.59**	**91.01**	89.56	91.24	89.79	91.08	**95.88**	**96.68**	87.17	88.93	**87.93**	89.83
GN13	86.75	88.40	87.38	88.87	87.31	89.10	87.36	88.84	94.06	95.00	85.68	87.60	85.20	87.19
LTDP	n/a	86.81	n/a	87.43	n/a	88.87	n/a	88.40	n/a	93.52	n/a	85.85	n/a	86.37
Mate	**89.03**	**90.73**	89.30	90.82	**90.09**	**91.92**	**90.28**	**91.68**	95.71	96.64	**87.86**	**89.87**	87.86	**89.89**
RBG	88.64	90.58	88.99	90.86	89.28	91.45	89.85	91.47	95.27	96.41	87.36	89.65	87.12	89.61
Redshift	88.60	90.19	88.96	90.46	89.11	90.90	89.63	90.99	95.36	96.22	87.14	88.99	87.27	89.31
SNN	85.35	87.08	86.13	87.78	86.00	87.92	86.17	87.74	93.47	94.64	83.50	85.74	84.29	86.50
spaCy	87.27	89.05	87.70	89.31	87.37	89.29	88.00	89.52	94.28	95.27	85.67	87.65	85.16	87.40
Turbo	87.05	88.70	87.58	89.04	88.34	90.02	87.95	89.33	94.39	95.36	85.91	87.93	85.66	87.70
Yara	88.90	90.53	89.40	90.89	89.72	91.42	90.00	91.41	95.41	96.32	87.35	89.19	87.55	89.61
Total	2211		1357		780		2326		1869		1366		1787	

Table 7: Parsing accuracy by genre.

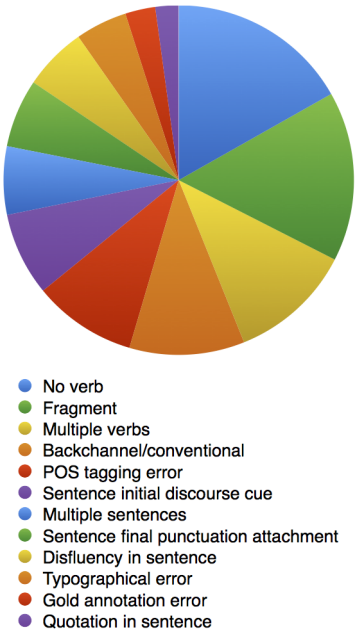


Figure 6: Common error types in erroneous trees.

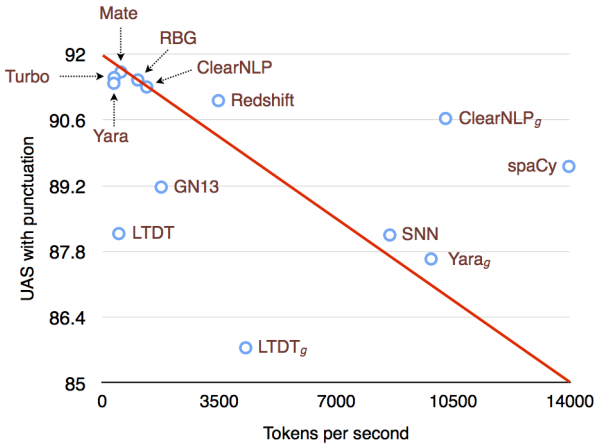


Figure 7: Speed with respect to accuracy.

6.6 Recommendations

Each of the transition-based parsers that was included in this evaluation can use varying beam widths to trade off speed vs. accuracy, and each parser has numerous other parameters that can be tuned. Notwithstanding all these variables, we can make some recommendations. Figure 7 illustrates the speed vs. accuracy tradeoff across the parsers. For highest accuracy (e.g. in dialog systems), Mate, RBG, Turbo, ClearNLP and Yara are good choices. For highest speed (e.g. in web-scale NLP), spaCy and ClearNLP$_g$ are good choices; SNN and Yara$_g$ are also good choices when accuracy is relatively not as important.

7 Conclusions and Future Work

In this paper we have: (a) provided a detailed comparative analysis of several state-of-the-art statistical dependency parsers, focusing on accuracy and speed; and (b) presented DEPENDABLE, a new web-based evaluation and visualization tool for analyzing dependency parsers. DEPENDABLE supports a wide range of useful functionalities. In the future, we plan to add regular expression search over parses, and sorting within results tables. Our hope is that the results from the evaluation as well as the tool will give non-experts in parsing better insight into which parsing tool works well under differing conditions. We also hope that the tool can be used to facilitate evaluation and be used as a teaching aid in NLP courses.

Supplements to this paper include the tool, the parse outputs, the statistical models for each parser, and the new set of dependency trees for OntoNotes 5 created using the ClearNLP dependency converter. We do recommend examining one's data and task before choosing and/or training a parser. Are non-projective parses likely or desirable? Does the data contain disfluencies, sentence fragments, and other "noisy text" phenomena? What is the average and standard deviation for sentence length and dependency length? The analyses in this paper can be used to select a parser if one has the answers to these questions.

In this work we did not implement an ensemble of parsers, partly because an ensemble necessarily entails complexity and/or speed delays that render it unusable by all but experts. However, our analyses indicate that it may be possible to achieve small but significant increases in accuracy of dependency parsing through ensemble methods. A good place to start would be with ClearNLP, Mate, or Redshift in combination with LTDP and Turbo, SNN or spaCy. In addition, it may be possible to achieve good performance in particular genres by doing "mini-ensembles" trained on general purpose data (e.g. WB) and genre-specific data. We leave this for future work. We also leave for future work the comparison of these parsers across languages.

It remains to be seen what downstream impact differences in parsing accuracy of 2-5% have on the goal task. If the impact is small, then speed and ease of use are the criteria to optimize, and here spaCy, ClearNLP$_g$, Yara$_g$ and SNN are good choices.

Acknowledgments

We would like to thank the researchers who have made available data (especially OntoNotes), parsers (especially those compared in this work), and evaluation and visualization tools. Special thanks go to Boris Abramzon, Matthew Honnibal, Tao Lei, Danqi Li and Mohammad Sadegh Rasooli for assistance in installation, trouble-shooting and general discussion. Additional thanks goes to the kind folks from the SANCL-SPMRL community for an informative discussion of evaluation and visualization tools. Finally, we would like to thank the three reviewers, as well as Martin Chodorow, Dean Foster, Joseph Le Roux and Robert Stine, for feedback on this paper.

References

Emily M. Bender, Dan Flickinger, Stephan Oepen, and Yi Zhang. 2011. Parser evaluation over local and non-local deep dependencies in a large corpus. In *Proceedings of EMNLP*.

Anders Björkelund et al. 2014. Introducing the IMS-Wrocław-Szeged-CIS entry at the SPMRL 2014 shared task: Reranking and morpho-syntax meet unlabeled data. In *Proceedings of the First Joint Workshop on Statistical Parsing of Morphologically Rich Languages and Syntactic Analysis of Non-Canonical Languages*.

Bernd Bohnet. 2010. Very high accuracy and fast dependency parsing is not a contradiction. In *Proceedings of COLING*.

Sabine Buchholz and Erwin Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of CoNLL*.

Danqi Chen and Christopher Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proceedings of EMNLP*.

Jinho D. Choi and Andrew McCallum. 2013. Transition-based Dependency Parsing with Selectional Branching. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, ACL'13, pages 1052–1062.

Jinho D. Choi and Martha Palmer. 2012a. Fast and Robust Part-of-Speech Tagging Using Dynamic Model Selection. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*, ACL'12, pages 363–367.

Jinho D. Choi and Martha Palmer. 2012b. Guidelines for the Clear Style Constituent to Dependency Conversion. Technical Report 01-12, Institute of Cognitive Science, University of Colorado Boulder, Boulder, CO, USA.

Ronan Collobert et al. 2011. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12:2493–2537.

Marie-Catherine de Marneffe and Christopher D. Manning. 2008. The Stanford typed dependencies representation. In *Proceedings of the COLING workshop on Cross-Framework and Cross-Domain Parser Evaluation*.

Yoav Goldberg and Joakim Nivre. 2013. Training deterministic parser with non-deterministic oracles. In *Proceedings of TACL*.

Gholamreza Haffari, Marzieh Razavi, and Anoop Sarkar. 2011. An ensemble model that combines syntactic and semantic clustering for discriminative dependency parsing. In *Proceedings of ACL-HLT*.

Jan Hajič et al. 2009. The CoNLL-2009 shared task: Syntactic and semantic dependencies in multiple languages. In *Proceedings of CoNLL*.

Matthew Honnibal, Yoav Goldberg, and Mark Johnson. 2013. A non-monotonic arc-eager transition system for dependency parsing. In *Proceedings of CoNLL*.

Liang Huang, Suphan Fayong, and Yang Guo. 2012. Structured perceptron with inexact search. In *Proceedings of the NAACL*.

Richard Johansson and Pierre Nugues. 2007. Extended constituent-to-dependency conversion for English. In *Proceedings of NODALIDA*.

Jonathan K. Kummerfeld et al. 2012. Parser showdown at the wall street corral: An empirical investigation of error types in parser output. In *Proceedings of EMNLP*.

Joseph Le Roux et al. 2012. DCU-Paris13 systems for the SANCL 2012 shared task. In *Proceedings of the First Workshop on Syntactic Analysis of Non-Canonical Language (SANCL)*.

Joseph Le Roux, Antoine Rozenknop, and Jennifer Foster. 2013. Combining PCFG-LA models with dual decomposition: A case study with function labels and binarization. In *Proceedings of EMNLP*.

Tao Lei, Yu Xin, Yuan Zhang, Regina Barzilay, and Tommi Jaakkola. 2014. Low-rank tensors for scoring dependency structures. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1381–1391, Baltimore, Maryland, June. Association for Computational Linguistics.

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of english: the Penn treebank. *Computational Linguistics*, 19(2):313–330.

André F. T. Martins, Miguel B. Almeida, and Noah A. Smith. 2013. Turning on the turbo: Fast third-order non-projective turbo parsers. In *Proceedings of the ACL*.

Ryan McDonald and Joakim Nivre. 2007. Characterizing the errors of data-driven dependency parsing models. In *Proceedings of EMNLP-CoNLL*.

Ryan McDonald and Joakim Nivre. 2011. Analyzing and integrating dependency parsers. *Computational Linguistics*, 37(1):197–230.

Makoto Miwa et al. 2010. A comparative study of syntactic parsers for event extraction. In *Proceedings of BioNLP*.

Jens Nilsson and Joakim Nivre. 2008. MaltEval: An evaluation and visualization tool for dependency parsing. In *Proceedings of LREC*.

Joakim Nivre et al. 2007. The CoNLL 2007 shared task on dependency parsing. In *Proceedings of CoNLL*.

Joakim Nivre, Laura Rimell, Ryan McDonald, and Carlos Gómez Rodríguez. 2010. Evaluation of dependency parsers on unbounded dependencies. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 833–841, Beijing, China, August. Coling 2010 Organizing Committee.

Stephan Oepen et al. 2014. SemEval 2014 Task 8: Broad-Coverage Semantic Dependency Parsing. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 63–72.

Slav Petrov and Ryan McDonald. 2012. Overview of the 2012 shared task on parsing the web. In *Proceedings of the First Workshop on Syntactic Analysis of Non-Canonical Language (SANCL) Shared Task*.

Slav Petrov, Pi-Chuan Chang, Michael Ringgaard, and Hiyan Alshawi. 2010. Uptraining for accurate deterministic question parsing. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 705–713, Cambridge, MA, October. Association for Computational Linguistics.

Sameer Pradhan et al. 2013. Towards robust linguistic analysis using OntoNotes. In *Proceedings of CoNLL*.

Mohammad Sadegh Rasooli and Joel R. Tetreault. 2015. Yara parser: A fast and accurate dependency parser. *CoRR*, abs/1503.06733.

Kenji Sagae and Alon Lavie. 2006. Parser combination by reparsing. In *Proceedings HLT-NAACL*.

Kenji Sagae and Jun'ichi Tsujii. 2010. Dependency parsing and domain adaptation with data-driven LR models and parser ensembles. In *Trends in Parsing Technology: Dependency Parsing, Domain Adaptation, and Deep Parsing*, pages 57–68. Springer.

Djamé Seddah et al. 2013. Overview of the SPMRL 2013 shared task: A cross-framework evaluation of parsing morphologically rich languages. In *Proceedings of the 4th Workshop on Statistical Parsing of Morphologically Rich Languages*.

Pontus Stenetorp et al. 2012. BRAT: A web-based tool for NLP-assisted text annotation. In *Proceedings of the EACL*.

Mihai Surdeanu et al. 2008. The CoNLL-2008 shared task on joint parsing of syntactic and semantic dependencies. In *Proceedings of CoNLL*.

Reut Tsarfaty, Joakim Nivre, and Evelina Andersson. 2011. Evaluating dependency parsing: Robust and heuristics-free cross-annotation evaluation. In *Proceedings of EMNLP*.

Ralph Weischedel et al. 2011. OntoNotes: A large training corpus for enhanced processing. In Joseph Olive, Caitlin Christianson, and John McCary, editors, *Handbook of Natural Language Processing and Machine Translation*. Springer.

Deniz Yuret, Laura Rimell, and Aydin Han. 2013. Parser evaluation using textual entailments. *Language Resources and Evaluation*, 47(3).