L95 Practical: Parser Evaluation and Comparison

Zhuoni Jie
Dept. of Computer Science and Technology
University of Cambridge
zj245@cam.ac.uk

Word Count: 4961

I. INTRODUCTION

A variety of parses have been developed for a variety of grammar formalisms, and comparing parsers with different grammars and different output is of great importance. Considerable progress has been made in accurate statistical parsing of texts, yielding rooted, hierarchical and relational representations of full sentences (e.g., [1], [2]). Many parsers have been made with systems based on large lexicalized probabilistic context-free like models trained on the Wall Street Journal (WSJ) subset of the Penn Treebank (PTB) [3]. Evaluation of these systems has been mostly in terms of metrics including attachment score for dependency parsers [4] and the PARSEVAL scheme [5] using tree similarity measures of labelled precision and recall and crossing bracket rate applied to section 23 of the WSJ PTB, which has now been criticised for not representing "real" parser quality [6][7]. As a result, infrequent yet semantically important construction types could be parsed with accuracies far below what one might expect.

This project researches into the syntactic parsing of natural languages, and does evaluation and comparison of parsing systems using MaltEval [32] with qualitative analysis. I evaluated the comparative accuracy of three unlexicalized statistical parsers on a small dataset of 25 sentences, summarized and compared the errors made by each parser, and made analysis regarding their processing speeds. Preprosessing issues and related concerns are also discussed.

II. PARSERS

This project focuses on three representative parsers: the Stanford parser [8], the Berkeley parser [9], and RASP (Robust Accurate Statistical Parsing) [10]. Owing largely to the Penn Treebank [3], the mainstream of data-driven parsing research has been dedicated to the phrase structure parsing. These parsers output Penn Treebank-style phrase structure trees, with function tags and empty categories stripped off. While most of the state-of-the-art parsers are based on Probablistic Context Free Grammars (PCFG), the parameterization of the probabilistic model of each parser varies. In this project, two of the three parsers are based on grammars automatically extracted from the PTB: the Stanford parser and the Berkeley Parser. The RASP parser uses a manually constructed grammar and a statistical parse selection component.

A. Stanford Parser

The Stanford parser is a statistical parser implemented in Java which modifies the original grammar rules to introduce semantics. It can be trained on annotated data, and applied to a number of languages such as English, Chinese, and Arabic.

There are different model implementations of the Stanford parser, as stated by Klein and Manning [11]. The default Stanford parser uses unlexicalized PCFG and can parse much more accurately than previously shown by making use of simple, linguistically motivated state splits, and outperforms early lexicalized PCFG models. This model provides good and fast parser for English, and has options for fine-tuning. It also has a factored model [12], which is a combined approach of lexicalized PCFG and dependency parse; and a shift-reduce model [13], which is based on PCFG with Recursive Neural Network (RNN) and w best accuracy. Socher et al. [14] describe a recent model of this parser. They introduce a Compositional Vector Grammar (CVG), which combines PCFGs with a syntactically untied RNN that learns syntactico-semantic, and compositional vector representations. The CVG helps in capturing the discrete categorisation of phrases into NP or PP while the RNN helps in capturing fine-grained syntactic and compositional-semantic information about phrases and words. The combination of the two helps finding the syntactic structure as well as capturing compositional semantic information, which gives the parser access to rich syntactico-semantic information in the form of distributional word vectors and computes compositional semantic vector representations for longer phrases.

B. Berkeley Parser

The Berkeley parser is a statistical parsing system implemented in Java for English and has been used in languages such as Chinese, German, Arabic, and Bulgarian. It is an implementation of smoothed unlexicalized PCFG parser whose non-terminals are refinements of the original treebank grammar obtained by an automatic splitmerge procedure.

The parameterization of this parser is optimized automatically by assigning latent variables to each nonterminal node and estimating the parameters of the latent variables by the Expectation-Maximization (EM) algorithm [15]. By using a split-and-merge strategy and beginning with the barest possible initial structure, the method reliably learns a PCFG that is remarkably good at parsing. Hierarchical split/merge

training enables the parser to learn compact but accurate grammars. Splitting provides a tight fit to the training data, while merging improves generalization and controls grammar size. In order to overcome data fragmentation and overfitting, the parameters were smoothed, which allows the parser to add a larger number of annotations, each specializing in only a fraction of the data, without overfitting the training set. The resulting grammar is automatically learned and human-interpretable. It shows most of the manually introduced annotations discussed by Klein and Manning [11], but also learns other linguistic phenomena. The above coarse-to-fine scheme in conjunction with the risk-appropriate parse selection methodology used in the Berkeley parser allows fast and accurate parsing, in multiple languages and domains. For training, a raw context-free treebank is needed. And for decoding, a final grammar along with coarsening maps is needed.

C. RASP (Robust Accurate Statistical Parsing)

RASP is a statistical feature-based parser that uses raw texts as inputs and outputs syntactic and parse analyses, such as parse trees, grammatical relations between lexical heads, or minimal recursion semantic analyses. RASP uses tag sequence grammar (TSG) [16], and the tagset utilized by the TSGs is based on the CLAWS tagset [17], as used in the Susanne Corpus [18]. It builds a dependency graph through unification operations performed during a phrase structure tree parsing process. Since RASP's first release, the use of TSG parsed text has been experimented on areas such as topic classification [19], sentiment classification [20], and question answering [21]. RASP is implemented as a series of modules written in C and Common Lisp, which are pipelined, working as a series of Unix-style filters. Its modular pipeline architecture whereby one module feeds the next one includes the following modules: tokenizer, Part of Speech (POS) and punctuation tagging, lemmatiser, parser/grammar, and a parse ranking model. Compared to its first public release in 2002 [22], the new version released in 2006 [10] includes a revised and more semantically-motivated output representation, an enhanced grammar and POS tagger lexicon, and a more flexible and semi-supervised training method for the structural parse ranking model. The current grammar outputs grammatical relations (GRs) based on a rule-to-rule encoding of GRs with PS rules, and can output other representations including weighted GRs, Grammar Rules, PTB-style (however, the tree topology will be more informative than Penn Treebank as RASP used the X-bar scheme), Shallow Phrasal, and Alias. Among the available outputs, the GRs are the most stable as a command line option in the distribution.

III. EXPERIMENTS

In this section, I describe the experimental setup for the evaluation and comparison, including dataset, pre-processing, post-processing, and evaluation.

A. Dataset

25 input sentences as provided in the practical handout are used in this project. These sentences have a minimum length (including punctuations) of 10 words, a maximum length of 56 words, and an average length of 28 words. I use the one-sentence-per-line format in the original data file. I also use the Shared Gold Standard annotation which were manually constructed by all the attendees of class L95 2018, and I did several modifications. The full original data can be found at the file directory: *smb://filer.cl.cam.ac.uk/userfiles/zj245/unix_home/L95/data*

B. Pre-Processing

Sometimes parsers can have problems with a variety of input. Common problematic cases include: long sentences; sentences with less traditional vocabulary; sentences with less traditional punctuation. These can make parsers make wrong analysis because of sentence boundary detection, length, complexity, or through spurious interactions between constituents. To help solve the first and third problems, I split the sentences where subsentences split by a semicolon or a colon can still be syntactically independent sentences. Sentences in direct quotations are also split by full stops. These manipulations result in 31 sentences for analysis. Samples are illustrated as in Figure 1 below.

To make the sentences suitable for input to parsers, and to compare their GRs output excluding the influence of POS tagging of each parser, I use the tokenization and POS input as notated in our Shared Gold Standard. Punctuations are seperated from adjacent words. Note that the Penn Treebank POS taggers are used directly as the one-sentence-per-line format for the Stanford parser. For the Berkeley parser, the one-word-per-line format is used, and the brackets are transformed into "-LRB-" for "(" and "-RRB-" for ")". For RASP, the Penn Treebank POS tags are converted and some of them are manually modified to the CLAWS2 tags as parser input. The format for each sentence is adopted to the requirement of RASP system (e.g., add "^_^" at the start of each sentence). Conversion samples are illustrated as in Fig. 1 below.

Due to the limited number of input sentences, and since both qualitative and quantitative analysis are expected, I ran all of the 31 sentences as parser input, although parsing is computationally expensive. I used default settings for the parsing models. The pre-processed data for each parser input can be found at the file directory: *smb://filer.cl.cam.ac.uk/userfiles/zj245/unix_home/L95/data*

C. Post-Processing

The three parsers do not output GRs in the same format. The Stanford parser can output formats in CoNLL [23], UD [24], PTB, and SD [25] styles. The Berkeley parser outputs formats in PTB-style, and always starts the tree with a ROOT node at the top, whereas the Stanford parser always surrounds the PTB tree with an extra set of parenthesis. RASP outputs formats in RASP and PTB styles. In this project, I also convert each parser's default output into

Original Data	Parser Inputs		
	Stanford parser	Berkeley parser	RASP
Kim and Sandy both broke up with their partners.	Kim/NNP and/CC Sandy/NNP both/RB broke/VBD up/RP with/IN their/PRP$ partners/NNS ./.	Kim NNP and CC Sandy NNP both RB broke VBD up RP with IN their PRP$ partners NNS . .	^_^ Kim_NP1 and_CC Sandy_NP1 both_DB2 broke_VVD up_RP with_IW their_APP$ partners_NN2 ._.

Fig. 1. A sample sentence for three parsers' input formats

CoNLL for evaluation and comparison. CoNLL format is the dependency tree format used in the 2006 and 2007 CoNLL shared tasks [23] [26] on dependency parsing. This representation format is obtained from PTB-style trees by applying constituent-to-dependency conversion [27].

Because of the wide variation in parser output representations, a method combining the use of Stanford conversion script, which converts PTB-style to GRs, with manual processing was performed to ensure that each parser output is converted correctly. The Stanford conversion program relies on function tags and empty categories of the original Penn Treebank. Although RASP can also output PTB-style trees, since it uses English CLAWS2 POS tagset and the X-bar scheme making the tree different in topology from PTB, it cannot be converted directly. I mannually constructed the CoNLL format of the RASP GRs. Fig. 2 shows the scheme for representation conversion. The post-processed data for each parser output can be found at the file directory: *smb://filer.cl.cam.ac.uk/userfiles/zj245/unix_home/L95/orig_output*

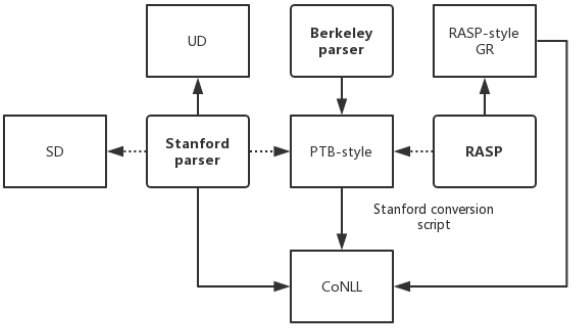

Fig. 2. The scheme for output representation conversion

Another thing worth mentioning is that I modified some relation names (e.g., changed names, added and merged some specifications of the relation type) for the evaluation using metrics. More specific modifications and examples will be explained in the following section.

D. Evaluation

There are several very useful tools for evaluating the output of parsers, including the *eval.pl* script [28] used in the CoNLL shared tasks, TedEval [31], ParsEval [30], tools embedded in parsers such as the Stanford parser and RASP, web-based tool such as DependAble [31], and MaltEval [32]. MaltEval is a software tool written in Java that combines quantitative evaluation and qualitative evaluation by outputing evaluation metrics, visualizing dependency structures, and highlighting errors for analyzing parsers. It is to a large extent adapted to the evaluation script *eval.pl*, as the functionality of MaltEval is essentially a superset of this script, and it is also more flexible and contains many additional features.

Here I use MaltEval for quantitative evaluation using our manually annotated gold standard and the full set processed as test data. All the data inputs are converted to CoNLL format. As aforementioned, we use UD in our Shared Gold standard. The gold standard data can be found at the file directory: *smb://filer.cl.cam.ac.uk/userfiles/zj245/unix_home/L95/data*

However, there are several inconsistencies in categorizing and naming rules for each parser's UD-style output. When using MaltEval, I encoded additional specifications of the relation type for some relations, merged some relations, and added the initial or underlying logical relation of the grammatical subject in constructions such as passive as needed, to make them consistent for comparison when carrying out the evaluation tasks.

Stanford parser's UD output is more detailed and contains subcategories for some UD relations. For example, the *compound* in UD are divided into *compound* and *compound:prt*, where *prt* denotes a phrasal verb particle. Similarly, the *nmod* in UD are *nmod*, *nmod:tmod*, *npmod*, and *nmod:poss* in Stanford output. I merged these kinds of subcategories according to relations in the original UD lists. Some relations also have different namings. For instance, Stanford parser uses *mwe* to denote the multi-word expression (modifier) relation, which is used for certain multi-word idioms that behave like a single function word (e.g. as well as), while UD uses *fixed* for words after the first one in multi-word

expressions. Additionally, the original Stanford parser output eliminated *punct* notations, and I manually added them for indexing and analyzing consistence.

Berkeley parser's output is originally PTB-style and converted to UD using Stanford conversion script. In this way, some grammatical relations are discarded in Berkeley's output (e.g., *compound* and *mod* s including *amod*, *nmod*, *advmod*, and *nummod*). For these words, Berkeley's converted output reflects their POS, such as *prt*, *nn*, and *poss*. These tags can only be mapped correctly if the lemma or the syntactic context is also known. The Stanford conversion script uses only tags (and features) as input; therefore the result is only an approximation of correct UD tags. In order to provide accurate and complete UD information, I did manual postprocessing, mapping the Berkeley output to UD tags if the Berkeley's tag result does not conflict with the Gold Standard's UD tag (e.g., *nn* can be reasonably mapped to *nmod* if the tree structure also permits). Berkeley's output also contains extra grammatical subject such as passives (e.g., *nsubjpass* and *auxpass*) while UD tags in our Gold Standard do not contain these information. In addition, Berkeley parser tags preposition and conjunction phrases (clauses) in different ways and with more details, specifying the specific prepositions and conjunctions. For example, the Gold Standard has: the old car broke down in*(9, case)* the*(9, det)* car*(9, nmod)* park*(4, nmod)*. And the Berkeley converted output has: the old car broke down in*(0, erased)* the*(9, det)* car*(9, nn)* park*(4, prep_in)*. I manually excluded these extra subjects, modified relation tagging rules, to make the tags consistent with our standard UD tags. Berkeley parser's converted output also contains some *dep* tags. A dependency is labeled as *dep* when the system is unable to determine a more precise dependency relation between two words. This may be because of a weird grammatical construction, a limitation in the Stanford Dependency conversion software, a parser error, or because of an unresolved long distance dependency. Berkeley output also contains some *erased* tags. When the CoNLL format is used with collapsed dependencies, words of the sentences which have been collapsed into the grammatical relations (such as prepositions and conjunctions) still appear in the list of words but are given an *erased* grammatical relation. I leave these *dep* tags as original, and map the corresponding *erased* tags to suited UD tags.

RASP's output is RASP-style GRs, and I mannually converted the GR results to the CoNLL format. I also made some modifications to make the output suitable for evaluation, and here are several sample issues. RASP uses *ncmod* as binary relations between non-clausal modifiers and their heads, while our Gold Standard uses taggings containing more information such as *amod*, *nmod*, *advmod*, and *nummod*. RASP also captures double object construction by having *obj2* tag, while UD does not have this. *ncmod prt* tag in RASP is equivalent to *compound* in UD. RASP also has different relation namings (e.g., *prep* in RASP is *case* in UD, and *iobj* in RASP is *obl* in UD). In addition, RASP has new tags containing sentence punctuation information (e.g., *ta*) and tags containing different subtypes (e.g., *ccomp* and *xcomp*) and non-binary relations that need conversion. RASP tag past participles as *passive* while UD tag them assigning more meanings to their functions such as *root*, *amod*, and *parataxis*. The tagging method regarding conjunctions is also different. RASP tags both components connected by conjunctions as *conj*, and does not specify conjunctions as *cc*, while UD tags only the latter one as *conj* and specifies the conjunctions. The conjunctions in RASP are tagged with GR of the core words in the conjunction phrases. RASP also tags possessive adjectives as *det*, while UD tags as *nmod*. Unlike Stanford parser and Berkeley parser, RASP does not tag *root* and some of the words in RASP do not output relations. What's more, some words may have more than one relations as dependents. I made relative modifications considering these issues in post-processing the RASP CoNLL file.

All the processed CoNLL files for evaluation can be found at the file directory: *smb://filer.cl.cam.ac.uk/userfiles/zj245/unix_home/L95/CoNLL_output*

IV. RESULTS

In this section, I present the detailed results for the experiments, including accuracy, error analysis, speed, and discussion.

A. Accuracy

The qualitative MaltEval results are shown in Fig. 3, where the accuracy for each construction is the percentage of Gold Standard dependencies recovered correctly. Here due to the limitation of data size and performance, I use the LA (LabelRight) Metric value for detailed reporting, and a token is counted as a hit if the dependency label is the same as in the gold-standard data. The Average row represents a macroaverage, which is calculated by taking the average of each measure for each individual relation and feature. I also use the LAS (BothRight) Metric and UAS (HeadRight) to report the overal parse accuracy. In LAS Metric value, a token is counted as a hit if both the head and the dependency label are the same as in the gold-standard data. In UAS Metric value, a token is counted as a hit if the head is the same as in the gold-standard data. The results are presented in Fig. 4.

Overall, the Stanford parser get the highest accuracy using UAS, LAS, and LA metrics, which means it has the best performance at detecting the relation labels and heads in our dataset. The Berkeley parser gets a relatively high LA accuracy score comparative to Stanford parser, but much lower UAS and LAS accuracy scores, indicating that it is good at detecting labels but comparatively weaker at detecting heads. This may be caused by Stanford's conversion script's limited ability to decode the Berkeley output's tree structure information, or by the parser's limited ability to solve weird grammar constructions or long distance dependencies. RASP has similar UAS and LAS accuracy scores compared to the Berkeley parser, but has lower LA accuracy score. This is probably because of RASP's incomplete detection of

Relation	Precision			Recall			F_1		
	Stanford	**Berkeley**	**RASP**	**Stanford**	**Berkeley**	**RASP**	**Stanford**	**Berkeley**	**RASP**
acl	0.866	**0.909**	0.8	0.688	0.625	**0.75**	0.759	0.741	**0.774**
advcl	0.444	0.667	**1**	**0.5**	**0.5**	0.375	0.471	**0.571**	0.545
advmod	**0.897**	0.842	0.721	**0.921**	0.842	0.816	**0.909**	0.842	0.765
amod	**0.978**	0.827	0.975	**0.936**	0.915	0.83	**0.957**	0.869	0.897
appos	0.333	0.5	**0.75**	0.143	0.143	**0.429**	0.2	0.222	**0.545**
aux	**0.957**	0.741	0.909	**1**	0.909	0.714	**0.978**	0.816	0.8
case	**1**	0.953	0.957	**0.97**	0.924	0.682	**0.985**	0.938	0.796
cc	0.931	**1**	0.909	**0.964**	0.857	0.714	**0.947**	0.923	0.8
ccomp	0.333	0.308	**0.364**	0.429	**0.571**	**0.571**	0.375	0.4	**0.444**
compound	0.667	**1**	0.895	**0.9**	0.25	0.85	0.766	0.4	**0.872**
conj	**0.786**	0.773	0.641	0.667	0.515	**0.758**	**0.721**	0.618	0.694
cop	**0.375**	0.333	1	**0.6**	**0.6**	0.2	**0.462**	0.429	0.333
csubj	0.5	0.5	0.5	1	1	1	0.667	0.667	0.667
dep	0.08	**0.097**	-	0.5	**0.75**	0	0.138	**0.171**	-
det	**1**	0.98	0.869	0.855	0.891	**0.964**	0.922	**0.933**	0.914
expl	1	1	1	**1**	**1**	0.5	1	1	0.667
fixed	0.8	1	1	1	0.75	1	0.889	0.857	**1**
mark	0.824	0.833	**1**	**0.778**	0.556	0.389	**0.8**	0.667	0.56
nmod	0.667	**0.676**	0.621	**0.806**	0.774	0.29	**0.73**	0.722	0.396
nsubj	**0.859**	0.806	0.786	**0.948**	0.931	0.759	**0.902**	0.864	0.772
nummod	**1**	0.875	0.8	0.778	0.778	**0.889**	**0.875**	0.824	0.842
obj	0.763	**0.767**	0.358	0.644	0.733	**0.756**	0.699	**0.75**	0.486
obl	-	1	1	0	**0.929**	0.357	-	**0.963**	0.526
parataxis	0	**1**	0.667	0	0.25	**0.5**	-	0.4	**0.571**
punct	1	1	1	0.861	0.899	**1**	0.925	0.947	**1**
root	0.871	0.867	**1**	**0.931**	0.897	0.138	**0.9**	0.881	0.242
xcomp	0.556	0.5	**0.8**	**0.714**	0.286	0.571	0.625	0.364	**0.667**
Average	0.711	0.769	0.820	0.723	0.706	0.622	0.744	0.696	0.676

Fig. 3. Accuracy (precision, recall, and F1 scores) of three parsers

Parser	UAS(%)	LAS(%)	LA(%)
Stanford parser	82.9	74.0	81.3
Berkeley parser	58.1	49.8	78.7
RASP	53.7	48.5	69.6

Fig. 4. Accuracy of three parsers for UAS, LAS, and LA metric values

GRs, where the grammar explicitly leaves some types of attachment decisions for post-processing.

Fig. 3 reports performance (Precision, Recall, and F1 scores) for all the reported relation labels. The difference between their Precision accuracies is not statistically significant (p ¿ 0.05). The dependency relations with the highest LA precision scores are *acl*, *amod*, *case*, *cc*, *det*, *fixed*, *mark*, *nummod*, and *root* (¿80%). Most of these relations have strong lexical clues, occur very often, or relatively short relations. Those with the lowest LA precision scores (¡50%) are *ccomp* and *csubj*, probably due to complex grammatical structures and relatively longer relations in sentences. Some relations occur very rare (e.g., *parataxis* and *expl*), and thus are not representative for ananlysis using the score. And the dependency relations with the biggest accuracy variations across parsers are *appos*, *compound*, *cop*, and *obj*.

Stanford parser does better on most relations, *advmod*, *amod*, *nsubj*, *case*, *aux*, and *cop* in particular; the Berkeley parser is better at *dep*, *obj*, and *obl*; and RASP is better at *ccomp*. Stanford parser seems to excel at short-distance dependencies, possibly because it uses more local features than the Berkeley parser, whose PCFG can capture longer-distance rules. The RASP seems to be better at capturing clausal components. Overall, all the three parsers do well in detecting *acl*, *advmod*, *amod*, *aux*, *case*, *cc*, and *det*, probably because these relations are of short distance and relatively fixed. These parsers generally perform worse at detecting *ccomp*, *dep*, and *xcomp*, possibly because these involves with more oblique, complex, and long relations in sentences.

B. Error Analysis

We now proceed to a more detailed error analysis based on the development sets. We are most interested in which dependency relations were computed with most errors, error types, and for which errors/accuracy varied between parsers. To find the main error types, I looked at the metric evaluation results and did visualization of tree structures and errors using MaltEval. I also explored by looking at some most frequent and most rarest labels in our sentence set, and see if they are correctly labeled. See the following figures for examples of some sentences highlighting errors for each parser.

For the Stanford parser, the most frequent error types are *nmod* (38 counts), *dep* (25 counts), *nsubj* (13 counts), *advmod* (10 counts), and *conj* (10 counts). The Stanford

parser tend to misclassify the head of *nmod* when coordinating conjunction exists; when sometimes cannot distinguish between oblique nominals (*obl*); when the many nouns coexist and their relationships in sentences are complex and cross-interactive. A sample is shown in Fig. 5. The Stanford parser has limitation in identifying some unusual punctuations (e.g., ";" and "-"), and sometimes these punctuations can separate syntactically independent sentences. Several punctuations are labeled as *dep*. The Stanford parser is also limited in identifying when proper nouns act as nominal modifier or object such as "HMM" in "takes as input a single HMM", and limited in dealing with syntactic structures in quotation marks and brackets such as *appos* and *compound* relations that involve interactions with words outside the quotation/bracket. Long and complex relations in quotation marks and brackets also cause false tags for many *nsubj* relations. For example, the Stanford parser wrongly marks $nsubj \mid negatives <= thank$ for "negatives (no, not), politeness markers (please, thank you)". Stanford parser also has issues when dealing with clausal components (*ccomp*) and *nsubj*, where grammatical structures become ambiguous and complex. For example, Stanford tag $nsubj \mid coders <= make$ for "help human coders make" while the correct tag is $ccomp \mid help => make$. *advmod* error sometimes happen with the apperance of markers (e.g., "when tagging languages" should be $mark \mid when <= tagging$ instead of $advmod \mid when <= tagging$) or conjunctions. For *conj*, Stanford parser sometimes wrongly identifies marks as conjunctions, and sometimes has problem dealing with noun clauses in conjunction structures. Some erroneous patterns are probably caused by the Stanford dependency representation rules, which leave out some syntactic information and require more heuristic rules to recover the correct relations.

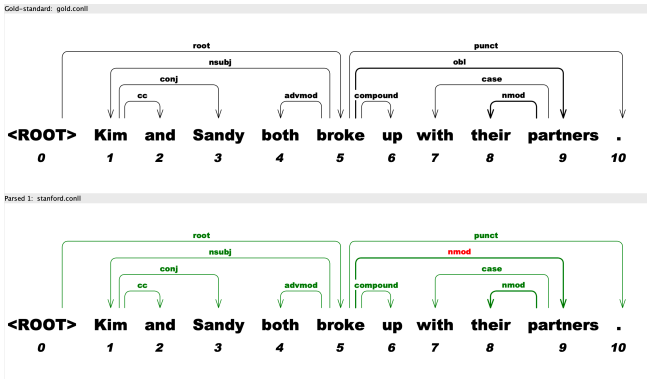

Fig. 5. A sample sentence when Stanford parser cannot distinguish between oblique nominal and nominal modifier

For the Berkeley parser, the most frequent error types are *case* (64 counts), *nmod* (40 counts), *dep* (30 counts), *nsubj* (29 counts), and *cc* (24 counts). Due to the PTB-to-CoNLL conversion rules of the Stanford conversion script, all the head of *case* and *cc* tags and many *obj* tags are directed to *root*, which is the main cause of the error frequencies of them. Berkeley parser cannot distinguish some compounds from noun modifiers, for example, "the Brown Corpus" is tagged as $compound \mid Brown <= Corpus$ but Berkeley identifies as $nmod \mid Brown <= Corpus$. Berkeley tags *dep* when encountering unusual words (e.g., "e.g.", and "Q = (q1, q2, ...qT)") and punctuations such as quotations (e.g., "the EM-trained 'pure HMM' tagger"). As shown in the sample sentence below, Berkeley is relatively good at identifying long-distance relations if some mapping rules in the Stanford conversion process can be improved.

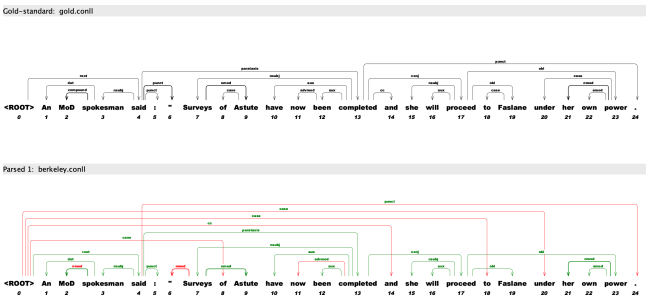

Fig. 6. A sample sentence showing Berkeley is relatively good at identifying long-distance relations if some mapping rules in the Stanford conversion process can be improved

For RASP, the most frequent error types are *obj* (68 counts), *unspecified* (66 counts), *case* (43 counts), *conj* (31 counts), *nsubj* (24 counts), *nmod* (20 counts), and *advmod* (18 counts). The main problem is that the RASP parser systematically fails to select the correct analysis for some SCFs with nouns and adjectives regardless of their context of occurrence. In the processed CoNLL files, RASP implements different rules to tag *obj* and *case* for prepositional phrases. For example, "in the car park" has a relation $case \mid in <= park$ which is tagged as $obj \mid in => park$. The *unspecified* tags are due to the limitation of RASP, where the grammar explicitly leaves some types of attachment decisions for post-processing. RASP sometimes also falsely identifies clausal modifier of noun (adjectival clause). For example, "Letters delivered on time" has $acl \mid Letters => delivered$ while this is tagged as $nsubj \mid Letters <= delivered$. RASP sometimes fails in identifying the correct head for *conj*. When coordinating phrases exist, RASP tags *nsubj* to the conjunction word, while the gold-standard tags to the first verb. When complex interactive sentence structure exists, the number of errors of modifier relations such as *nmod* and *advmod* increases.

Generally speaking, most local labels have a consistently high accuracy. As sentence length increases, more loose joining relations, deeper GR arcs, and more complex punctuations, the number of parse errors increases. The errors can be grouped into three categories as Nivre et al. did [27]:

- A *global error* is one where the parser completely fails to build the relevant clausal structure, and when a global error occurs, it is usually meaningless to further classify the error, which means that this category excludes the other two.
- An *Arg error* is one where the parser has constructed the relevant clausal structure but fails to find the *Arg*

relation (in the simple and complex cases), or the set of surrounding *Arg* relations needed to infer an implicit *Arg* relation (in the indirect case).

- A *Link error* is one where the parser fails to find the crucial *Link* relation.

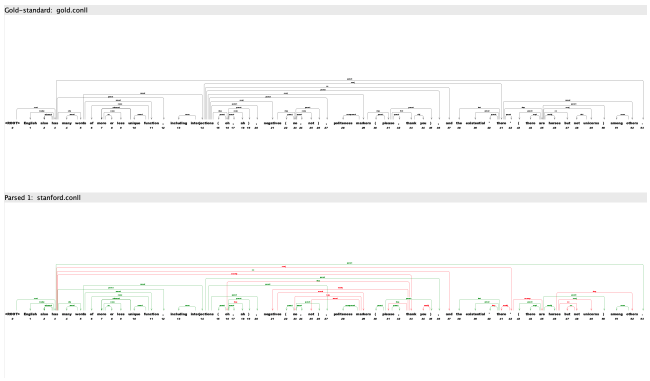

Fig. 7. A sample sentence illustrating errors with long relation distances and deep arcs

C. Speed

Parser	Load Time			Run Time		
	real	user	sys	real	user	sys
Stanford parser	0.85	2.20	0.12	17.71	28.69	0.37
Berkeley parser	6.13	11.10	0.53	6.83	13.46	0.55
RASP	0.67	0.54	0.10	1.08	0.49	0.24

Fig. 8. The load time and parse time of the three parsers

The timing experiments were run on a 64-bit machine with 2.6 GHz Intel Core i7 processor and 16 GB 2400 MHz DDR4 RAM, and used the unix *time* command to time each run. Here I report the load time and the parse time of the parsers. The load time is estimated by running the shortest sentence in our dataset through the parser and timing the execution, averaging three executions. Each run time result is obtained by averaging three identical parsing implementations on the full dataset, using one thread. Note that RASP takes longer time when loading the first parse, and then becomes stable. Here I exclude the first RASP parse time when computing the average parsing time.

We can see RASP is the quickest among the three parsers, about 58 times quicker than Stanford parser and 27 times quicker than Berkeley parser (regarding user run time). RASP is also the quikest in loading the parser. This is probably because of factors including its C implementation, classification algorithm, grammars, and complexity of the parsing algorithm. The Stanford and Berkeley constituency parsers use CockeYoungerKasami (CYK) algorithm which has computational complexity of $O(n^3|G|)$, where n is the length of the sentence and $|G|$ is the number of grammar productions, and RASP uses first-order hidden markov model (HMM) with a complexity of $O(n^2T)$, where T is the number of tags. The Stanford parser has shorter load time but longer run time compared to the Berkeley parser. I report the Stanford parser's pasing speed for each sentence as in Fig. 6. As expected, with increased sentence lengths, parsing speed decreases remarkably.

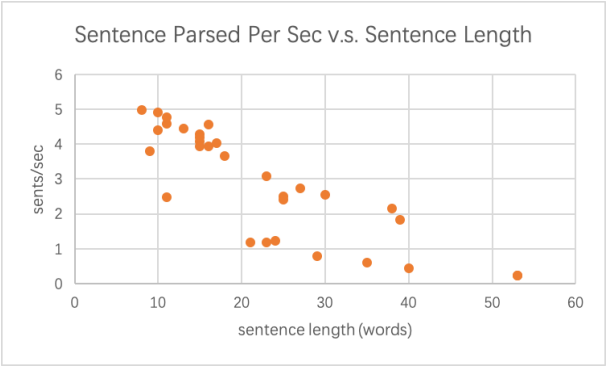

Fig. 9. Parse time for each sentence of the Stanford parser

D. Discussion

To properly evaluate the quality of syntactic parsing, we need to choose an appropriate metric that will provide the most relevant and representative information. Apart from LA, LAS, and UAS metrics, there are other standard ones to deal with complicated evaluation issues and considerations. For example, the Leaf-Ancestor Evaluation is conducted by assigning a score to every word in a sentence. The lineage of each word in the output tree (i.e. the sequence of nodes one has to pass to get to the word, starting from the root) is compared to the lineage of the same word in the gold tree, and the score is calculated as the minimum edit distance. PARSEVAL is a previously popular metric, where a node in a tree is considered to be correct if there is the same node in the corresponding gold tree. Similarity is defined by the indices of the substring that the node dominates and the label of the node. The problem of standard Parseval is that it counts nodes as the same regardless of the underlying structure they dominate. To eliminate some of the problems in PARSEVAL evaluation, Ringger et al. [34] have proposed Maximal Projections of Heads (MPHs) suitable for evaluating and comparing both probabilistic and rule-based parsers.

Universal dependencies are not the only popular dependency representation, and we may also consider other formats for analysis. The process to obtain CoNLL-style relations changed and eliminated some original syntactic and structural information produced by each parser, and causes inconsistency and errors when manually processing the data and obtain CoNLL-style dependency trees from parser output. In addition, although modified some parts, there are still false relations tagged in our Gold Standard. More accurate gold standard is desired for performing our evaluation tasks.

Our results suggest that if accuracy is of primary concern, then Stanford parser should be preferred; however, Berkeley parser offers a nice trade-off between accuracy and speed,

and is better at detecting distant relations. The RASP is the fastest option, and getting good relation name detection results. If excluding the undetected ones and further modifying the RASP-to-CoNLL rules, the RASP-detected relations are reasonably accurate. Fortunately, recent research has come with much headroom for improving parsing accuracy, including a tunable margin parameter C for the classifier, richer feature sets, and ensemble models.

V. CONCLUSION

A contribution of this project has been to describe the difficulties and issues associated with cross-formalism parser comparison, with a considerable amount of pre- and post-processing with regard to different formats and schemes. I evaluated and compared the accuracy of three state-of-the-art parsers on a sentence dataset, identified parsing error types and traits, and analyzed their parsing speeds.

More work is needed to enable more accurate, informative, objective, and systematic comparison of a wider range of different parsers, probably on a larger dataset. I have also experienced difficulties for relational evaluation schemes and realized that combining individual scores for different relations and features with qualitative analysis is desired for parsing system comparisons. More meaningful and concrete comparison and classification methods also remain to be explored.

REFERENCES

[1] Toutanova, Kristina, et al. "Parse disambiguation for a rich HPSG grammar." First Workshop on Treebanks and Linguistic Theories (TLT2002), 253-263. Stanford InfoLab, 2002.

[2] Kaplan, Ronald M., et al. Speed and accuracy in shallow and deep stochastic parsing. PALO ALTO RESEARCH CENTER CA, 2004.

[3] Marcus, Mitchell P., Mary Ann Marcinkiewicz, and Beatrice Santorini. "Building a large annotated corpus of English: The Penn Treebank." Computational linguistics 19.2 (1993): 313-330.

[4] Buchholz, Sabine, and Erwin Marsi. "CoNLL-X shared task on multilingual dependency parsing." Proceedings of the tenth conference on computational natural language learning. Association for Computational Linguistics, 2006.

[5] Black, Ezra, et al. "A procedure for quantitatively comparing the syntactic coverage of English grammars." Speech and Natural Language: Proceedings of a Workshop Held at Pacific Grove, California, February 19-22, 1991. 1991.

[6] Briscoe, Ted, et al. "Relational evaluation schemes." Proceedings of the beyond PARSEVAL Workshop at the 3rd International Conference on Language Resources and Evaluation. 2002.

[7] Carroll, John, Guido Minnen, and Ted Briscoe. "Corpus annotation for parser evaluation." arXiv preprint cs/9907013 (1999).

[8] Manning, Christopher, et al. "The Stanford CoreNLP natural language processing toolkit." Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations. 2014.

[9] Petrov, Slav, et al. "Learning accurate, compact, and interpretable tree annotation." Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics. Association for Computational Linguistics, 2006.

[10] Briscoe, Ted, John Carroll, and Rebecca Watson. "The second release of the RASP system." Proceedings of the COLING/ACL on Interactive presentation sessions. Association for Computational Linguistics, 2006.

[11] Klein, Dan, and Christopher D. Manning. "Accurate unlexicalized parsing." Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1. Association for Computational Linguistics, 2003.

[12] Klein, Dan, and Christopher D. Manning. "Fast exact inference with a factored model for natural language parsing." Advances in neural information processing systems. 2003.

[13] Chen, Danqi, and Christopher Manning. "A fast and accurate dependency parser using neural networks." Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). 2014.

[14] Socher, Richard, John Bauer, and Christopher D. Manning. "Parsing with compositional vector grammars." Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). Vol. 1. 2013.

[15] Matsuzaki, Takuya, Yusuke Miyao, and Jun'ichi Tsujii. "Probabilistic CFG with latent annotations." Proceedings of the 43rd annual meeting on Association for Computational Linguistics. Association for Computational Linguistics, 2005.

[16] Briscoe, Ted, and Nick Waegner. "Robust stochastic parsing using the inside-outside algorithm." Proc. of the AAAI Workshop on Probabilistic-Based Natural Language Processing Techniques. 1992.

[17] Garside, Roger. "The CLAWS word-tagging system." (1987).

[18] rey Sampson, Geo. "English for the Computer." The SUSANNE Cor (1995).

[19] Bennett, Paul N., and Jaime Carbonell. "Detecting action-items in e-mail." Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval. ACM, 2005.

[20] Khan, S. "Negation in sentiment analysis: employing contextual valence shifters to enhance classification accuracy", Proceedings of the ACL-Coling06 (2006), submitted.

[21] Leidner, Jochen L., et al. "QED: The Edinburgh TREC-2003 Question Answering System." TREC. 2003.

[22] Briscoe, Ted, and John A. Carroll. "Robust Accurate Statistical Annotation of General Text." LREC. 2002.

[23] Buchholz, Sabine, and Erwin Marsi. "CoNLL-X shared task on multilingual dependency parsing." Proceedings of the tenth conference on computational natural language learning. Association for Computational Linguistics, 2006.

[24] McDonald, Ryan, et al. "Universal dependency annotation for multilingual parsing." Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers). Vol. 2. 2013.

[25] De Marneffe, Marie-Catherine, et al. "Universal Stanford dependencies: A cross-linguistic typology." LREC. Vol. 14. 2014.

[26] Nivre, Joakim, et al. "The CoNLL 2007 shared task on dependency parsing." Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL). 2007.

[27] Johansson, Richard, and Pierre Nugues. "Extended constituent-to-dependency conversion for English." (2007).

[28] https:github.comallenaiallennlpblobmasterallennlptoolssrl-eval.pl

[29] Tsarfaty, Reut, Joakim Nivre, and Evelina Ndersson. "Evaluating dependency parsing: robust and heuristics-free cross-nnotation evaluation." Proceedings of the Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics, 2011.

[30] Harrison, Philip, et al. "Evaluating syntax performance of parser grammars." Proceedings of the Natural Language Processing Systems Evaluation Workshop, Berekely, CA, June 1991. 1991.

[31] Choi, Jinho D., Joel Tetreault, and Amanda Stent. "It depends: Dependency parser comparison using a web-based evaluation tool." Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers). Vol. 1. 2015.

[32] Nilsson, Jens, and Joakim Nivre. "MaltEval: an Evaluation and Visualization Tool for Dependency Parsing." LREC. 2008.

[33] Nivre, Joakim, et al. "Evaluation of dependency parsers on unbounded dependencies." Proceedings of the 23rd International Conference on Computational Linguistics. Association for Computational Linguistics, 2010.

[34] Ringger, Eric, et al. "Linguistically informed statistical models of constituent structure for ordering in sentence realization." COLING 2004: Proceedings of the 20th International Conference on Computational Linguistics. 2004.