A Fast, Effective, Non-Projective, Semantically-Enriched Parser

Stephen Tratz and **Eduard Hovy**
Information Sciences Institute
University of Southern California
Marina del Rey, California 90292
{stratz,hovy}@isi.edu

Abstract

Dependency parsers are critical components within many NLP systems. However, currently available dependency parsers each exhibit at least one of several weaknesses, including high running time, limited accuracy, vague dependency labels, and lack of non-projectivity support. Furthermore, no commonly used parser provides additional shallow semantic interpretation, such as preposition sense disambiguation and noun compound interpretation. In this paper, we present a new dependency-tree conversion of the Penn Treebank along with its associated fine-grain dependency link types and a parser that is, to the best of our knowledge, the first dependency parser capable of parsing more than 75 sentences per second at over 93% accuracy. We explain how a non-projective extension to shift-reduce parsing can be incorporated into non-directional easy-first parsing. The parser performs well when evaluated on the standard test section of the Penn Treebank, outperforming several popular open source dependency parsers.

1 Introduction

Parsers are critical components within many natural language processing (NLP) systems, including systems for information extraction, question answering, machine translation, recognition of textual entailment, summarization, and many others. Unfortunately, currently available dependency parsers suffer from at least one of several weaknesses including high running time, limited accuracy, vague dependency labels, and lack of non-projectivity support. Furthermore, few parsers include any sort of additional semantic interpretation, such as interpretations for prepositions, possessives, or noun compounds.

In this paper, we describe 1) a new dependency conversion (Section 3) of the Penn Treebank (Marcus, et al., 1993) along with the associated dependency label scheme, which is based upon the Stanford parser's popular scheme (de Marneffe and Manning, 2008), 2) a fast, accurate dependency parser with non-projectivity support (Section 4) and 3) additional integrated semantic annotation modules for automatic preposition sense disambiguation and noun compound interpretation (Section 5). We show how Nivre's (2009) swap-based reordering technique for non-projective shift-reduce-style parsing can be integrated into the non-directional easy-first framework of Goldberg and Elhadad (2010) to add support for non-projectivity. We report the results of our parsing experiments on the standard test section of the PTB, providing comparisons with parser implementations from MALTPARSER (Nivre et al., 2006), MSTPARSER (McDonald et al., 2005; McDonald and Pereira, 2006), and Goldberg and Elhadad's (2010) open source implementation.

The parsing evaluation results show that the parser is substantially more accurate than Goldberg and Elhadad's original implementation, with fairly similar overall speed. Furthermore, our results show that Stanford-granularlity dependency labels can be learned by modern dependency parsing systems when using our Treebank conversion, unlike the Stanford conversion, for which Cer et al. (2010) show that this isn't the case.

The optional semantic annotation modules also perform well, with the preposition sense disambiguation module exceeding the accuracy of the pre-

vious best reported result for fine-grained preposition sense disambiguation (85.6% vs Hovy et al.'s (2010) 84.8%) and the noun compound interpretation system performing similarly to an earlier version described by Tratz and Hovy (2010).

2 Background

The NLP community has recently seen a surge of interest in dependency parsing, with several CoNLL shared tasks focusing on it (Buchholz and Marsi, 2006; Nivre et al., 2007). One of the main advantages of dependency parsing is the relative ease with which it can handle non-projectivity[1]. Additionally, since each word is linked directly to its head via a link that, ideally, indicates the syntactic dependency type, there is no difficulty in determining either the syntactic head of a particular word or the syntactic relation type, whereas these issues often arise when dealing with constituent parses[2].

Unfortunately, most currently available dependency parsers produce relatively vague labels or, in many cases, produce no labels at all. While the Stanford fine-grain dependency scheme (de Marneffe and Manning, 2008) has proven to be popular, recent experiments by Cer et al. (2010) using the Stanford conversion of the Penn Treebank indicate that it is difficult for current dependency parsers to learn. Indeed, the highest scoring parsers produced using the MSTPARSER (McDonald and Pereira, 2006) and MALTPARSER (Nivre et al., 2006) parsing suites achieved only 78.8 and 81.1 labeled attachment F1, respectively. This contrasted with the much higher performance obtained using a constituent-to-dependency conversion approach with an accurate, but much slower, constituency parser such as the Charniak and Johnson (2005) and Berkeley (Petrov et al., 2006) parsers, which achieved 89.1 and 87.9 labeled F1 scores, respectively.

Though there are many syntactic parsers than can reconstruct the grammatical structure of a text, there are few, if any, accurate and widely accepted systems that also produce shallow semantic analysis of the text. For example, a parser may indicate that, in the case of 'ice statue', 'ice' modifies 'statue' but will not indicate that 'ice' is the *substance* of the statue. Similarly, a parser will indicate which words a preposition connects but will not give any semantic interpretation (e.g., 'the boy *with* the pirate hat' $\rightarrow$ *wearing or carrying*, 'wash *with* cold water' $\rightarrow$ *means*, 'shave *with* the grain' $\rightarrow$ *in the same direction as*). While, in some cases, it may be possible to use the output from a separate system for this purpose, doing so is often difficult in practice due to a wide variety of complications, including programming language differences, alternative data formats, and, sometimes, other parsers.

[1] A tree is non-projective if the sequence of words visited in a left-to-right, depth-first traversal of the sentence's parse tree is different than the actual word order of the sentence.

[2] These latter two issues are not problems for constituent parses with binarized output and functional tags.

3 Dependency Conversion

3.1 Relations and Structure

Most recent English dependency parsers produce one of three sets of dependency types: unlabeled, some variant of the coarse labels used by the CoNLL dependency parsing shared-tasks (Buchholz and Marsi, 2006; Nivre et al., 2007) (e.g., ADV, NMOD, PMOD), or Stanford's dependency labels (de Marneffe and Manning, 2008). Unlabeled dependencies are clearly too impoverished for many tasks. Similarly, the coarse labels of the CoNLL tasks are not very specific; for example, the same relation, NMOD, is used for determiners, adjectives, nouns, participle modifiers, relative clauses, etc. that modify nouns. In contrast, the Stanford relations provide a more reasonable level of granularity.

Our dependency relation scheme is similar to Stanford's basic scheme but has several differences. It introduces several new relations including *ccinit* "initial coordinating conjunction", *cleft* "cleft clause", *combo* "combined term", *extr* "extraposed element", *infmark* "infinitive marker 'to' ", *objcomp* "object complement", *postloc* "posterior location modifier", *sccomp* "clausal complement in so-that construction", *vch* "verbal chain" and *whadvmod* "wh- adverbial modifier". We leave out the *nsubjpass*, *csubjpass*, and *auxpass* relations of Stanford's because adding them up front makes learning more difficult and the fact that a *nsubj*, *csubj*, or *aux* is passive can easily be determined from the final tree. Instead of *aux* relations, we use *vch* links; conversion of these to Stanford-style *aux* dependencies is also trivial as a post-processing step.[3]The

abbrev	abbreviation	*csubjpass*	clausal subject (passive)	pobj	prepositional object
acomp	adjectival complement	det	determiner	poss	possessive
advcl	adverbial clause	dobj	direct object	possessive	possessive marker
advmod	adverbial modifier	**extr**	extraposed element	**postloc**	post-modifying location
agent	'by' agent	expl	'there' expletive	preconj	pre conjunct
amod	adjectival modifier	**infmark**	infinitive marker ('to')	predet	predeterminer
appos	appositive	infmod	infinite modifier	prep	preposition
attr	attributive	iobj	indirect object	prt	particle
aux	auxillary	mark	subordinate clause marker	punct	punctuation
auxpass	auxillary (passive)	measure	measure modifier	purpcl	purpose clause
cleft	cleft clause	neg	negative	quantmod	quantifier modifier
cc	coordination	nn	noun compound	rcmod	relative clause
ccinit	initial CC	nsubj	nominal subject	*rel*	relative
ccomp	clausal complement	*nsubjpass*	nominal subject (passive)	**sccomp**	clausal complement of 'so'
combo	combination term	num	numeric modifier	tmod	temporal modifier
compl	complementizer	number	compound number	**vch**	verbal chain
conj	conjunction	**objcomp**	object complement	**whadvmod**	wh- adverbial
cop	copula complement	parataxis	parataxis	xcomp	clausal complement w/o subj
csubj	clausal subject	partmod	participle modifier		

Table 1: Dependency scheme with differences versus basic Stanford dependencies highlighted. Bold indicates the relation does not exist in the Stanford scheme. Italics indicate the relation appears in Stanford's scheme but not ours.

attr dependency is excluded because it is redundant with the our *cop* relation due to different handling of copula, and the dependency scheme does not have an *abbrev* label because this information is not provided in the Penn Treebank. The dependency scheme along with differences with Stanford highlighted is given in Table 1.

In addition to using a slightly different set of dependency names, we treat a handful of relations, notably *cop*, *conj*, and *cc* differently. These are illustrated by Figure 1.

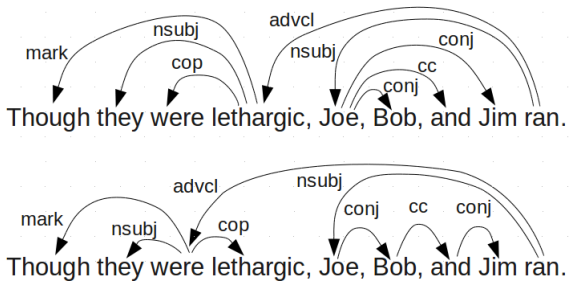

Figure 1: Example comparing Stanford's (top) handling of copula and coordinating conjunctions with ours (bottom).

[3]The parsing system includes an optional script that can convert *vch* arcs into *aux* and *auxpass* and the subject relations into *csubjpass* and *nsubjpass*.

3.2 Conversion Process

A three-step process is used to convert the Penn Treebank (Marcus, et al., 1993) from constituent parses into dependency trees labeled according to our dependency scheme. The first step is to apply the noun phrase structure patch created by Vadas and Curran (2007), which adds structure to the otherwise flat noun phrases (NPs) of the Penn Treebank (e.g., '(metal soup pot cover)' would become '(metal (soup pot) cover)'). The second step is to apply a version of Johansson and Nugues' (2007) constituent-to-dependency converter with some head-finding rule modifications; these rules are provided in Table 2. Finally, an additional script converts the intermediate output into our dependency scheme.

This dependency conversion has several advantages to it. Using the modified head-finding rules for Johansson and Nugues' (2007) converter results in fewer buggy trees than were present in the CoNLL shared tasks, including fewer trees in which words are headed by punctuation marks. For sections 2–21, there are far fewer generic *dep*/*DEP* relations (2765) than with the Stanford conversion (34,134) or the CoNLL 2008 shared task conversion (23,811). Also, the additional conversion script contains various rules for correcting part-of-speech (POS) errors using the syntactic structure as well as additional

(WH)?NP|NX|NML|NAC	**FW|NML**|NN* JJR **#|$** CD|**FW QP** JJ|**NAC** JJS **PRP ADJP RB[SR] VBG|DT|WP**
	RB NP-ϵ **S|SBAR|UCP|PP SINV|SBARQ|SQ UH VP|NP VB|VBP**
ADJP|JJP	NNS QP NN $|**#** JJ VBN VBG (AD|J)JP **ADVP** JJR NP|NML JJS DT FW RBR RBS SBAR RB
ADVP	**RB|RBR|JJ|JJR** RBS FW ADVP TO CD IN NP|NML JJS NN
PRN	S* **VP NN*|NX|NML NP** W* PP|IN ADJP|JJ ADVP RB **NAC VP INTJ**
QP	$|# NNS NN **CD** JJ RB DT NCD QP IN **CC** JJR JJS
SBARQ	SQ S SBARQ **SINV** FRAG
SQ	VBZ VBD VBP VB MD *-PRD SQ VP **FRAG X**
UCP	**[QNVP]P|S*|UCP|NML|PR[NT]|RRC|NX|NAC|FRAG|INTJ|AD[JV]P|LST|WH*|X**
VP	VBD|AUX VBN MD VBZ VB VBG VBP VP **POS** *-PRD ADJP JJ NN NNS NP|NML
WHADJP	CC **JJ** WRB ADJP
WHADVP	CC WRB|**RB**
X	**[QNVP]P|S*|UCP|NML|PR[NT]|RRC|NX|NAC|FRAG|INTJ|AD[JV]P|LST|WH*|X|CONJP**
LST	LS **: DT|NN|SYM**

Figure 2: Modified head-finding rules. Underline indicates that the search is performed in a left-to-right fashion instead of the default right-to-left order. NML and JJP are both products of Vadas and Curran's (2007) patch. Bold indicates an added or moved element; for the original rules, see the paper by Johansson and Nugues (2007).

rules for some specific word forms, mostly common words with inconsistent taggings. Many of these changes cover part-of-speech problems discussed by Manning (2011), including VBD/VBN, VBZ/NNS, NNP/NNPS, and IN/WDT/DT issues. In total, the script changes over 9500 part-of-speech tags, with the most common change being to change preposition tags (IN) into adverb tags (RB) for cases where there is no prepositional complement/object. The top fifteen changes are given in Table 2. The conversion script also contains a variety of additional rules for fixing erroneous trees, primarily for cases where one or more POS tags were incorrect and, as such, the initial dependency conversion was flawed. A quick, manual inspection suggested that the vast majority of these changes are accurate.

In sections 2–21, the number of sentences with one or more words dependent on non-projective arcs is 3245, or about 8.1% of the dataset.

4 Parser

4.1 Algorithm

Our parsing approach is based upon the *non-directional easy-first* algorithm recently presented by Goldberg and Elhadad (2010). Their original algorithm behaves as follows. For a sentence of length n, the algorithm performs a total of n steps. In each step, one of the unattached tokens is added as a child to one of its current neighbors and is then removed from the list of unprocessed tokens. When only one token remains unprocessed, it is designated as the root. Provided that only a constant number of potential attachments need to be re-evaluated after each step, which is the case if one restricts the context for feature generation to a constant number of neighboring tokens, the algorithm can be implemented to run in $O(n \log n)$. However, since only $O(n)$ dot products must be calculated by the parser and these have a large constant associated with them, the running time will rival $O(n)$ parsers for any reasonable n, and, thus, a naive $O(n^2)$ implementation will be nearly as fast as a priority queue implementation in practice.[4]

The algorithm has a couple potential advantages over standard shift-reduce style parsers. The first advantage is that performing easy actions first may make the originally difficult decisions easier. The second advantage is that performing parse actions in a more flexible order than left-to-right/right-to-left shift-reduce parsing reduces the chance of error propagation.

Unfortunately, the original algorithm does not support non-projective trees. To extend the algorithm to support non-projective trees, we introduce *move-right* and *move-left* operations similar to the stack-to-buffer *swaps* proposed by Nivre (2009) for shift-reduce style parsing. Thus, instead of attaching a token to one of its neighbors at each step, the algorithm may instead decide to *move* a token

[4] See Goldberg and Elhadad (2010) for more explanation.

Original	New	# of changes
IN	RB	1128
JJ	NN	787
VBD	VBN	601
RB	IN	462
VBN	VBD	441
NN	JJ	409
NNPS	NNP	405
IN	WDT	388
VBG	NN	223
DT	IN	220
RB	JJ	214
VB	VBP	184
NN	NNS	169
RB	NN	157
NNS	VBZ	148

Table 2: Top 15 part-of-speech tag changes performed by the conversion script.

past one of its neighbors. Provided that no node is allowed to be moved past a token in such a way that a previous *move* operation is undone, there can be at most O(n^2) *moves* and the overall complexity becomes O($n^2 \log n$). While theoretically slower, this has a limited impact upon actual parsing times in practice, especially for languages with relatively fixed word order such as English.[5] Goldberg and Elhadad's (2010) original implementation only supports unlabeled dependencies; however, it is simple enough to add support for labeled dependencies by treating each dependency label as a specific type of *attach* operation (e.g., *attach-as-nsubj*), which is the method used in this implementation. Pseudocode for the non-directional easy-first algorithm with non-projective support is given in Algorithm 1.

input : w_1 ... w_n, #the sentence
m, #the model
k, #the context width
$actions$, #the list of parse actions
ϕ, #the feature generator
output: $tree$ #a collection of dependency arcs
$working$ = `copyOf(`s`);`
$stale$ = `copyOf(`s`);`
$cache$; #cache of action scores
while |$working$| > *1* **do**
  for $w \in stale$ **do**
    for $act \in actions$ **do**
      $cache$[w,act] = `score(`*act, ϕ(w,...), m*`);`
    $stale$.`remove(`w`);`
  $best = \underset{a \in actions \& valid(a), w \in working}{\arg\max} cache[w, a]$
  if `isMove(`$best$`)` **then**
    idx =
      $working$.`index(getTokenToMove(`$best$`))`;
    $working$.`move(`idx, `isMoveLeft(`$best$`)` ?
    -1 : 1);
  else
    arc = `createArc(`$best$`);`
    $tree$.`add(`arc`);`
    $index$ = $working$.`index(getChild(`arc`))`;
    $working$.`remove(`$index$`);`
  for $i \in$ *-k,...,k* **do**
    $stale$.`add(`$working$.`get(`$index$+i`))`;
return $tree$

Algorithm 1: Modified version of Goldberg and Elhadad's (2010) Easy-First Algorithm with non-projective support.

4.2 Features

One of the key aspects of the parser is the complex set of features used. The feature set is based off the features used by Goldberg and Elhadad (2010) but has a significant number of extensions. Various feature templates are specifically designed to produce features that help with several syntactic issues including preposition attachment, coordination, adverbial clauses, clausal complements, and relative clauses. Unfortunately, there is insufficient space in this paper to describe them all here. However, a list of feature templates will be provided with the parser download.

Several of the feature templates use unsupervised word clusters created using the Brown et al. (1992) hierarchical clustering algorithm. The use of this algorithm was inspired by Koo et al. (2008), who used the top branches of the cluster hierarchy as features. However, unlike Koo et al. (2008), we do not restrict ourselves to the top 4-6 branches of the cluster hierarchy and instead use the fine-grained cluster identifiers. The parser utilizes 175 word clusters created using the New York Times corpus (Sandhaus, 2008); the ideal number of clusters was not thoroughly investigated.

4.3 Training

The parsing model is trained using a variant of the structured perceptron training algorithm used in the original Goldberg and Elhadad (2010) implementa-

[5] See Nivre (2009) for more information on the effect of reordering operations on parse time.

tion to train a model for determining the parse actions. The general idea of the algorithm is to iterate over the sentences and, whenever the model predicts an incorrect action, update the model weights. Following Goldberg and Elhadad, parameter averaging is used to reduce overfitting.

Our implementation varies slightly from that of Goldberg and Elhadad (2010). The difference is that, at any particular step for a given sentence, we continue to update the weight vector as long as *any* invalid action is scored higher than *any* valid action, not just the highest scoring valid action; unfortunately, this change significantly slowed down the training process. In early experiments, this change produced a slight improvement in accuracy though it also slowed training significantly. In later experiments using additional feature templates, this change ceased to have any notable impact on the overall accuracy, but it was kept anyway. [6]

The oracle used to determine whether a *move* operation is considered legal during the training phase similar to Nivre et al.'s (2009) improved oracle based upon *maximal projective subcomponents*. For training purposes, *move* actions were only considered valid either if no other action was valid or if the token to be moved already had all its children attached and moving it caused it to be adjacent to its parent. This fits with Nivre et al.'s (2009) intuition that it is best to delay swapping as long as possible.

4.4 Speed Enhancements

To enhance the speed for practical use, the parser uses constraints based upon the POS tags of the adjacent word pairs to eliminate invalid dependencies from even being evaluated. A relation is only considered between a pair of words if such a relation was observed in the training data between a pair of words with the same parts-of-speech (with the exception of the generic *dep* dependency, which is permitted between any POS tag pair). Early experiments utilizing similar constraints showed an improvement in parsing speed of about 16% with no significant impact on accuracy, regardless of whether the constraints were enforced during training.

[6] See Goldberg and Elhadad (2010) for more description of the general training procedure.

4.5 Evaluation

The following data split was used for the experiments: sections 2–21 for training, 22 for development, and 23 for testing.

For part-of-speech (POS) tagging, we used an in-house SVM-based POS tagger[7] inspired by the work of Giménez and Márquez (2004). The training data was tagged in a 10-fold fashion; each fold was tagged using a tagger trained from the nine remaining folds. The development and test sections were tagged by an instance of the tagger trained using the entire training set. The full details of the POS tagger are outside the scope of this paper; it is included with the parser download.

The final parser was trained for 33 iterations, which is the point at which its performance on the development set peaked. One test run was performed with non-projectivity support disabled in order to see what the impact of the *move* operations would be on the parser's overall performance; also, since the parsers used for comparison had no access to the unsupervised word clusters, an additional copy of the parser was trained with each word treated as belonging to the same cluster to facilitate a more fair comparison.

Seven different dependency parsing models were trained for comparison using the following open source parsing packages: Goldberg and Elhadad's (2010)'s *non-directional easy-first* parser, MALTPARSER (Nivre et al., 2006), and MSTPARSER (McDonald and Pereira, 2006) [8]. The model trained using Goldberg and Elhadad's (2010) easy-first parser serves as something of a baseline. The four parsing MALTPARSER models used the *arc-eager*, *arc-standard*, *stack-eager*, and *stack-lazy* algorithms. One of the MSTPARSER models used the Chu-Liu-Edmonds maximum spanning tree approach, and the other used the Eisner (1996) algorithm with second order features and a non-projective rewriting post-processing step.

Unfortunately, it is not possible to directly compare the parser's accuracy with most popular constituent parsers, such as the Charniak (2000) and Berkeley (Petrov et al., 2006; Petrov and Klein, 2007) parsers [9], both because they do not pro-

[7] 97.42% accuracy on traditional POS evaluation (Penn Treebank WSJ sections 22-24).

System	Arc Accuracy		Perfect Sentences		Non-Proj Arcs	
	Labeled	Unlabeled	Labeled	Unlabeled	Labeled	Unlabeled
THIS WORK	**92.1 (93.3)**	**93.7 (94.3)**	**38.7 (43.2)**	**46.2 (48.6)**	**65.7 (69.3)**	**67.7 (71.3)**
THIS WORK$_{no\ clusters}$	91.8 (93.1)	93.4 (94.0)	38.2 (42.3)	45.5 (47.3)	**65.7** (??.?)	67.3 (??.?)
THIS WORK$_{moves\ disabled}$	91.8 (92.9)	93.3 (93.9)	37.2 (41.2)	44.2 (46.2)	20.3 (21.1)	21.1 (21.9)
NON-DIR EASY FIRST	*	91.2 (92.0)	*	37.8 (39.4)	*	15.1 (16.3)
EISNER$^{\dagger}{}_{MST}$	90.9 (92.2)	92.8 (93.5)	32.1 (35.6)	40.6 (42.3)	62.5 (65.3)	63.7 (66.9)
CHU-LIU-EDMONDS$_{MST}$	90.0 (91.2)	91.8 (92.5)	28.4 (31.3)	35.0 (36.4)	62.9 (65.3)	64.1 (66.5)
ARC-EAGER$_{Malt}$	89.8 (91.1)	91.3 (92.1)	31.6 (34.2)	37.4 (38.5)	19.5 (19.5)	20.3 (19.9)
ARC-STANDARD$_{Malt}$	88.3 (89.5)	89.7 (90.4)	31.4 (34.1)	36.1 (37.3)	13.1 (12.0)	13.9 (12.7)
STACK-EAGER$_{Malt}$	90.0 (91.2)	91.5 (92.3)	34.5 (37.5)	40.4 (41.9)	51.8 (53.8)	53.8 (55.4)
STACK-LAZY$_{Malt}$	90.4 (91.7)	91.9 (92.8)	34.8 (37.7)	40.6 (42.5)	61.8 (63.3)	63.3 (65.3)
CHARNIAK‡	*	93.2	*	43.5	*	32.3
BERKELEY‡	*	93.3	*	43.6	*	34.3

Table 3: Parsing results for section 23 of the Penn Treebank (punctuation excluded). Results in parentheses were produced using gold POS tags. †Eisner (1996) algorithm with non-projective rewriting and second order features. ‡Results not directly comparable; see text. *Labeled dependencies not available/comparable.

duce functional tags for subjects, direct objects, etc., which are required for the final script of the constituent-to-dependency conversion routine, and because they determine part-of-speech tags in conjunction with the parsing. However, it is possible to compute an approximate unlabeled accuracy score by training the parsers on the NP-patched (Vadas and Curran, 2007) version of the data and then running the test output through just the first conversion script, that is, the modified version of Johansson and Nugues's (2007) converter.

The results of the experiment are given in Table 3, including accuracy for individual arcs, non-projective arcs only[10], and full sentence match. Punctuation is excluded in all the result computations. To evaluate the impact of part-of-speech tagging error, results for parsing using the gold standard part-of-speech tags are also included.

.1

We also measured the speed of the parser on the various sentences in the test collection. For reasonable sentence lengths, the parser scales quite well. A graph depicting the relation between sentence length and parsing time is presented in Figure 4.

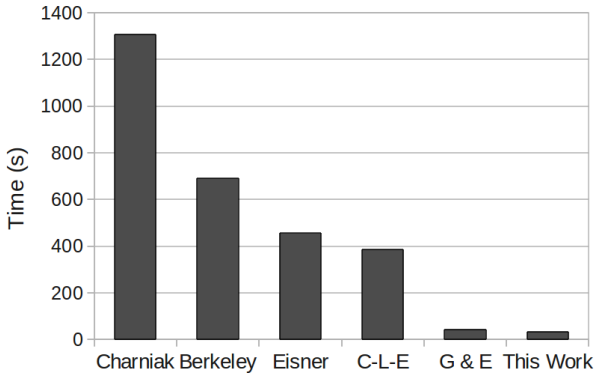

Figure 3: Parse times for the different parsers.[12](PTB section 23; PC with 2.4Ghz Q6600 processor).

4.5.1 Results Discussion

The parser performs very well overall, achieving 92.1% labeled and 93.7% unlabeled accuracy, about 2.5% higher than the original easy-first implementation of Goldberg and Elhadad (2010). Furthermore, the parser runs substantially faster than most of the other parses, processing over 75 sentences per second. Not surprisingly, the results for non-projective arcs are substantially lower than the results for all arcs.

The 93.2% and 93.3% accuracy scores achieved by the Charniak and Berkeley parsers is not too different from the 93.7% achieved by the new parser,

[8]Versions 1.4.1, 0.4.3b, and 0.2, respectively

[8]Versions 1.1 and 05Aug16, respectively

[9]To determine whether an arc is non-projective, we used the following heuristic. Traverse the sentence in a depth-first search, starting from the imaginary root node and pursuing child arcs in order of increasing absolute distance from their parent. If an arc being traversed is found to cross a previously traversed arc, it is marked as non-projective.

but, of course, it is important to remember that these scores are not directly comparable.

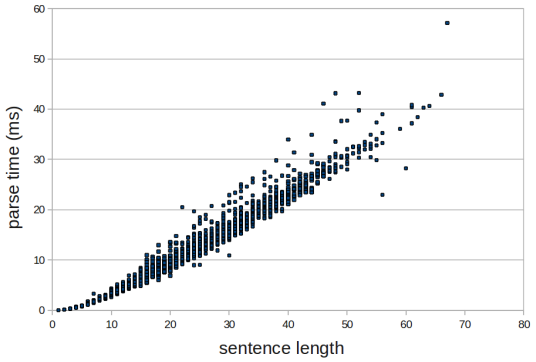

Figure 4: Sentence length versus parse time. Median times for 5 runs over section 23.

5 Shallow Semantic Annotation

To create a more informative parse, the parser includes four optional modules, a preposition sense disambiguation (PSD) system, a noun compound interpretation system, a work-in-progress *'s*-possessive interpretation system, and a PropBank-based semantic role labeling system[13]. The PSD system is a newer version of the system described by Tratz and Hovy (2009) and Hovy et al. (2010); it achieves 85.6% accuracy on the SemEval-2007 fine-grain PSD task (Litkowski and Hargraves, 2007), which is a statistically significant ($p<=0.05$; upper-tailed z test) increase over the previous best reported result for this dataset, Hovy et al.'s (2010) 84.8%. The noun compound interpretation system is a newer version of the system described by Tratz and Hovy (2010) with similar accuracy (79.1% accuracy using 10-fold cross-validation). The possessive interpretation system currently achieves over 80.0% accuracy, but the annotation scheme, automatic classifier, and dataset are all still under active development. The PropBank-based semantic role labeling system achieves 86.8 combined F_1 measure for automatically-generated parse trees (calculated over both predicate disambiguation and argument/adjunct identification and labeling); this score is not directly comparable to any previous work due to differences in the data conversion, including conversion of the parse trees, but is most similar to the CoNLL shared task work (Surdeanu et al., 2008; Hajič et al., 2009). Taken together, these optional modules enable the parsing system to produce substantially more informative output than a traditional parser.

[10]Lack of space prohibits a sufficiently thorough discussion of these individual components in this paper, but more information will be provided with the system download.

[11]MALTPARSER ran substantially slower than the other parsers, perhaps due to its use of polynomial kernels, and is not shown.

6 Related Work

Non-projectivity. There are two main approaches used in recent NLP literature for handling non-projectivity in parse trees. The first is to use an algorithm, like the one presented in this paper, that has inherent support for non-projective trees. Examples of this include the Chu-Liu-Edmonds' approach for maximum spanning tree (MST) parsing (McDonald et al., 2005) and Nivre's (2009) swap-based reordering method for shift-reduce parsing. The second approach is to create an initial projective parse and then apply transformations to introduce non-projectivity into it. Examples of this include McDonald and Pereira's (2006) rewriting of projective trees produced by the Eisner (1996) algorithm, and Nivre and Nilsson's (2005) pseudo-projective approach that creates projective trees with specially marked arcs that are later transformed into non-projective dependencies.

Descriptive dependency labels. While most recent dependency parsing research has used either vague labels, such as those of the CoNLL shared tasks, or no labels at all, some descriptive dependency label schemes exist. By far the most prominent of these is the Stanford typed dependency scheme (de Marneffe and Manning, 2008). Another descriptive scheme that exists, but which is less widely used in the NLP community, is the one used by Tapanainen and Järvinen's parser (1997). Unfortunately, the Stanford dependency conversion of the Penn Treebank has proven difficult to learn for current dependency parsers (Cer et al., 2010), and there is no publicly available dependency conversion according to Tapanainen and Järvinen's scheme.

Faster parsing. While the fastest reasonable parsing algorithms are the $O(n)$ shift-reduce algo-

rithms, such as Nivre's (2003) algorithm and an expected linear time dynamic programming approach presented by Huang and Sagae (2010), a few other fast alternatives exist. Goldberg and Elhadad's (2010) easy-first algorithm is one such example. Another example, is Roark and Hollingshead's (2009) work that uses chart constraints to achieve linear time complexity for constituency parsing.

Effective features for parsing. A variety of work has investigated the use of more informative features for parsing. This includes work that integrates second and even third order features (McDonald et al., 2006; Carreras, 2007; Koo and Collins, 2010). Also, some work has incorporated unsupervised word clusters as features, including that of Koo et al. (2008) and Suzuki et al. (2009), who utilized unsupervised word clusters created using the Brown et al. (1992) hierarchical clustering algorithm.

Semantically-enriched output. The 2008 and 2009 CoNLL shared tasks (Surdeanu et al., 2008; Hajič et al., 2009), which required participants to build systems capable of both syntactic parsing and Semantic Role Labeling (SRL) (Gildea and Jurafsky, 2002), are the most notable attempts to encourage the development of parsers with additional semantic annotation. These tasks relied upon PropBank (2005) and NomBank (2004) for the semantic roles. A variety of other systems have focused on FrameNet-based (1998) SRL instead.

7 Conclusion

In this paper, we presented a parser that is fast, accurate, supports non-projective trees, and provides rich output, including not only informative dependency labels but also additional semantic annotation for prepositions, possessives, and noun compound relations. We showed how the easy-first algorithm of Goldberg and Elhadad (Goldberg and Elhadad, 2010) can be extended to support non-projective trees by adding *move* actions similar to the Nivre's (2009) swap-based reordering for shift-reduce parsing, and described a new high-quality dependency tree conversion of the Penn Treebank (Marcus, et al., 1993) along with our labeled dependency scheme.

We demonstrated that the parser achieves high labeled and unlabeled accuracy of 92.1% and 93.7%, respectively. The 93.7% result represents a 2.5% increase over accuracy of Goldberg and Elhadad's (2010) implementation. The optional preposition sense disambiguation (PSD) module achieved 85.6% accuracy on the SemEval-2007 PSD task, exceeding the previous best published result of 84.8% by a statistically significant margin, while the noun compound interpretation module achieved 79.1% accuracy on Tratz and Hovy's (2010) dataset, similar to their results.

We showed that our parser is very fast, being capable of processing section 23 of the Penn Treebank in just over 30 seconds (a rate of over 75 sentences per second). The experimental results show that dependency parsers generally perform well on our PTB conversion, demonstrating that well-known dependency parsing algorithms can produce Stanford-granularity labels with high accuracy when using our dependency conversion of the Penn Treebank, which does not appear to be the case for the Stanford conversion according to the findings of Cer et al. (2010).

8 Future Work

In the future, we would like to extend this work in several ways. We would like to add support for traces of WH- words and also change our handling of coordinating conjunctions to treat the coordinating conjunction as the head.

To aid future NLP research work, the conversion and parsing code will be released for download from http://www.isi.edu .

Acknowledgements

We would like to thank Richard Johansson for providing us with the code for the *pennconverter* consituent-to-dependency converter. We would also like to thank Dirk Hovy and Anselmo Peñas for many valuable dicussions and suggestions.

References

Collin Baker, Charles J. Fillmore and John B. Lowe. 1998. The Berkeley FrameNet Project. In *Proceedings of the 17th international conference on Computational linguistics*

Ken Barker and Stan Szpakowicz. 1998. Semi-automatic recognition of noun modifier relationships. In *Pro-*

ceedings of the 17th international conference on Computational linguistics

Adam L. Berger, Vincent J. Della Pietra, and Stephen A. Della Pietra. 1996. A maximum entropy approach to natural language processing. In *Computational Linguistics* 22(1):39–71

Thorsten Brants and Alex Franz. 2006. Web 1T 5-gram Corpus Version 1. LDC2006T13. Linguistic Data Consortium, Philadelphia.

Peter F. Brown, Peter V. deSouza, Robert L. Mercer, Vincent J. Della Pietra, and Jenifer C. Lai. 1992. Class-Based n-gram Models of Natural Language. *Computational Linguistics* 18(4):467–479.

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

Xavier Carreras. 2007. Experiments with a Higher-Order Projective Dependency Parser. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*.

Daniel Cer, Marie-Catherine de Marneffe, Daniel Jurafsky, and Christopher D. Manning. 2010. Parsing to Stanford Dependencies: Trade-offs between speed and accuracy. In *7th International Conference on Language Resources and Evaluation (LREC 2010)*.

Eugene Charniak. 2000. A Maximum-Entropy-Inspired Parser. In *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-find-grained *n*-best parsing and discriminative reranking. In *Proceedings of the 43rd ACL*.

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*.

Jason Eisner. 1996. Three New Probabilistic Models for Dependency Parsing: An Exploration. In *Proceedings of the 16th COLING* pages 340-345. Association for Computational Linguistics.

Christiane Fellbaum. 1998. WordNet: An Electronic Lexical Database. MIT Press.

Daniel Gildea and Daniel Jurafsky. 2002. Automatic labeling of semantic roles. Computational Linguistics. 28(3):245–288.

Jesús Giménez and Lluís Márquez 2004. SVMTool: A General POS Tagger Generator Based on Support Vector Machines. In *Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC'04)*.

Yoav Goldberg and Michael Elhadad. 2010. An Efficient Algorithm for Easy-First Non-Directional Dependency Parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the ACL*, pages 742–750.

Yoav Goldberg and Michael Elhadad. 2009. The CoNLL-2009 Shared Task: Syntactic and Semantic Dependencies in Multiple Languages. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task*, pages 1–18.

Dirk Hovy, Stephen Tratz, and Eduard Hovy. 2010. What's in a Preposition?—Dimensions of Sense Disambiguation for an Interesting Word Class. In *Proceedings of COLING*.

Liang Huang and Kenji Sagae. 2010. Dynamic Programming for Linear-Time Shift-Reduce Parsing. In *Proceedings of ACL 2010*.

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

Terry Koo, Xavier Carreras, and Michael Collins. 2008. Simple Semi-supervised Dependency Parsing. In *Proceedings of ACL-08*, pages 595–603.

Terry Koo and Michael Collins. 2010. Efficient Third-order Dependency Parsers. In *Proceedings of ACL*.

Ken Litkowski and Orin Hargraves. 2005. The Preposition Project. In *Proceedings of the Second ACL-SIGSEM Workshop on the Linguistic Dimensions of Prepositions and their Use in Computational Linguistics Formalisms and Applications*.

Ken Litkowski and Orin Hargraves. 2007. SemEval-2007 Task 06: Word-Sense Disambiguation of Prepositions. In *Proceedings of the 4th International Workshop on Semantic Evaluations*.

Christopher D. Manning. 2011. Part-of-Speech Tagging from 97% to 100%: Is It Time for Some Linguistics? In *Proceedings of the 12th International Conference on Intelligent Text Processing and Computational Linguistics (CICLing 2011)*.

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.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005. Non-Projective Dependency Parsing Using Spanning Tree Algorithms. In *Proceedings of HLT-EMNLP*.

Ryan McDonald and Fernando Pereira. 2006. Online Learning of Approximate Dependency Parsing Algorithms. In *Proceedings of EACL*.

Adam Meyers, Ruth Reeves, Catherine Macleod, Rachel Szekely, Veronika Zielinska, Brian Young and Ralph Grishman. 2004. The NomBank Project: An Interim Report. In *Proceedings of the NAACL/HLT Workshop on Frontiers in Corpus Annotation*.

Joakim Nivre. 2009. Non-Projective Dependency Parsing in Expected Linear Time. In *Proceedings of the 47th Annual Meeting of the ACL and the 4th IJCNLP of the AFNLP*, pages 351-359.

Joakim Nivre. 2003. An Efficient Algorithm for Projective Dependency Parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT)*, pages 149–160.

Joakim Nivre, Marco Kuhlmann, and Johan Hall. 2009. An Improved Oracle for Dependency Parsing with Online Reordering. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT)*.

Joakim Nivre, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 2007. The CoNLL 2007 shared task on dependency parsing. In *Proceedings of EMNLP-CoNLL*.

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 and Jens Nilsson. 2005. Pseudo-projective dependency parsing. In *Proceedings of ACL-2005*.

Martha Palmer, Daniel Gildea, and Paul Kingsbury. 2005. The Proposition Bank: An Annotated Corpus of Semantic Roles. In *Computational Linguistics*. 31(1):71–106.

Slav Petrov and Dan Klein. 2007. Improved Inference for Unlexicalized Parsing. In *Proceedings of HLT-NAACL 2007*.

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

Brian Roark and Kristy Hollingshead. 2009. Linear complexity context-free parsing pipelines via chart constraints. In *Proceedings of HLT-NAACL*.

Evan Sandhaus. 2008. The New York Times Annotated Corpus. Linguistic Data Consortium, Philadelphia.

Mihai Surdeanu, Richard Johansson, Adam Meyers, Lluís Màrquez, and Joakim Nivre. 2008. The CoNLL-2008 shared task on joint parsing of syntactic and semantic dependencies. In *Proceedings of the Twelfth Conference on Computational Natural Language Learning*.

Jun Suzuki, Hideki Isozaki, Xavier Carrerras, and Michael Collins. 2009. An Empirical Study of Semi-supervised Structured Conditional Models for Dependency Parsing. In *Proceedings of EMNLP*.

Pasi Tapanainen and Timo Järvinen. 1997. A non-projective dependency parser. In *Proceedings of the fifth conference on applied natural language processing*.

Stephen Tratz and Dirk Hovy. 2009. Disambiguation of Preposition Sense using Linguistically Motivated Features. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Student Research Workshop and Doctoral Consortium*.

Stephen Tratz and Eduard Hovy. 2010. A Taxonomy, Dataset, and Classifier for Automatic Noun Compound Interpretation. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*.

David Vadas and James R. Curran. 2007. Adding Noun Phrase Structure to the Penn Treebank. In *Proceedings of ACL*.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical Dependency Analysis With Support Vector Machines. In *Proceedings of 8th International Workshop on Parsing Technologies*, pages 195–206.