IWPT 2015

14th International Conference on Parsing Technologies

Proceedings of the Conference

July 22–24, 2015
Bilbao, Spain

Order copies of this and other ACL proceedings from:

Association for Computational Linguistics (ACL)
209 N. Eighth Street
Stroudsburg, PA 18360
USA
Tel: +1-570-476-8006
Fax: +1-570-476-0860
acl@aclweb.org

ISBN 978-1-941643-98-3

Preface

Welcome to the 14th International Conference on Parsing Technologies (IWPT 2015), in the beautiful city of Bilbao! The conference is the 14th in the series of biennial meetings on parsing technologies organised by SIGPARSE, the Special Interest Group on Natural Language Parsing of the Association for Computational Linguistics (ACL). This year, IWPT is co-located with the 6th Workshop on Statistical Parsing of Morphologically Rich Languages (SPMRL 2015).

IWPT 2015 received 26 submissions (18 long and 8 short). All papers were reviewed by four members of the programme committee (see the following pages), and finally ten long and five short papers were accepted for presentation. The conference has two invited talks, by Slav Petrov (Google) and Mirella Lapata (University of Edinburgh). There are three social events: a visit to local bars for drinks and pintxos, a guided visit of the world famous Guggenheim museum, and the conference dinner at the renowned Etxanobe restaurant. The SPMRL workshop is interleaved with IWPT on Thursday 23rd July, starting after the morning coffee break.

We are indebted to the reviewers, for in-depth and constructive feedback, to all authors, for their collaboration in preparing camera-ready manuscrips, and to Murhaf Fares (University of Oslo) for assistance in compiling the proceedings. We would like to thank the conference sponsors (the University of the Basque Country, and the Government of the Basque Country) for their generous financial support. On behalf of the local arrangements committee we wish you a pleasant stay in Bilbao, and we hope you enjoy the conference.

John Carroll (General Chair),
Koldo Gojenola (Organizing Chair), and
Stephan Oepen (Programme Chair)

Organizers:

John Carroll, University of Sussex, United Kingdom (General Chair)
Koldo Gojenola, University of the Basque Country, Spain (Organizing Chair)
Stephan Oepen, University of Oslo, Norway (Programme Chair)

Programme Committee:

Željko Agić, Denmark
Miguel Ballesteros, Spain
Guntis Barzdins, Latvia
Anders Björkelund, Germany
Bernd Bohnet, United Kingdom
Aoife Cahill, United States
Jennifer Foster, Ireland
Filip Ginter, Finland
Carlos Gómez-Rodríguez, Spain
Angelina Ivanova, Norway
Richard Johansson, Sweden
Marco Kuhlmann, Sweden
Sadao Kurohashi, Japan
Sandra Kübler, United States
André F. T. Martins, Portugal
David McClosky, United States
Paola Merlo, Switzerland
Yusuke Miyao, Japan
Lluís Màrquez, Qatar
Mark-Jan Nederhof, United Kingdom
Joakim Nivre, Sweden
Woodley Packard, United States
Barbara Plank, Denmark
Wolfgang Seeker, Germany
Vijay Shanker, United States
Weiwei Sun, China
Mihai Surdeanu, United States
Ivan Titov, Netherlands
Gertjan van Noord, Netherlands
Luke Zettlemoyer, United States
Yi Zhang, Germany
Yue Zhang, Singapore
Lilja Øvrelid, Norway

Table of Contents

Conference Programme

Wednesday, July 22

9:30–10:30	*Towards Universal Syntactic Processing of Natural Language* Slav Petrov
11:00–11:30	*Domain Adaptation for Dependency Parsing via Self-Training* Juntao Yu, Mohab Elkaref and Bernd Bohnet
11:30–12:00	*Combining Active Learning and Partial Annotation for Domain Adaptation of a Japanese Dependency Parser* Daniel Flannery and Shinsuke Mori
12:00–12:30	*Incorporating Complementary Annotation to a CCGbank for Improving Derivations for Japanese* Sumire Uematsu and Yusuke Miyao
14:00–14:30	*Dependency Parsing with Graph Rewriting* Bruno Guillaume and Guy Perrier
14:30–15:00	*Semantic Parsing for Textual Entailment* Elisabeth Lien and Milen Kouylekov
15:00–15:30	*A Framework for Procedural Text Understanding* Hirokuni Maeta, Tetsuro Sasada and Shinsuke Mori
16:00–16:20	*Suitability of ParTes Test Suite for Parsing Evaluation* Marina Lloberes, Irene Castellón and Lluís Padró
16:20–16:40	*Coordination-Aware Dependency Parsing (Preliminary Report)* Akifumi Yoshimoto, Kazuo Hara, Masashi Shimbo and Yuji Matsumoto
16:40–17:00	*MSTParser Model Interpolation for Multi-Source Delexicalized Transfer* Rudolf Rosa and Zdeněk Žabokrtský

Thursday, July 23

9:20–9:50 *Non-Deterministic Oracles for Unrestricted Non-Projective Transition-Based Dependency Parsing*
Anders Björkelund and Joakim Nivre

9:50–10:10 *Enhancing the Inside-Outside Recursive Neural Network Reranker for Dependency Parsing*
Phong Le

10:10–10:30 *Maximising Spanning Subtree Scores for Parsing Tree Approximations of Semantic Dependency Digraphs*
Natalie Schluter

Friday, July 24

Large-scale Semantic Parsing as Graph Matching
Mirella Lapata

11:00–11:30 *CKY Parsing with Independence Constraints*
Joseph Irwin and Yuji Matsumoto

11:30–12:00 *Dependency Parsing with Compression Rules*
Pablo Gamallo

12:00–12:30 *Stacking or Supertagging for Dependency Parsing – What's the Difference?*
Agnieszka Faleńska, Anders Björkelund, Özlem Çetinoğlu and Wolfgang Seeker

Domain Adaptation for Dependency Parsing via Self-training

Juntao Yu[1], **Mohab Elkaref**[1], **Bernd Bohnet**[2]
[1] University of Birmingham, Birmingham, UK
[2]Google, London, UK
[1]{j.yu.1, m.e.a.r.elkaref}@cs.bham.ac.uk, [2]bohnetbd@google.com

Abstract

This paper presents a successful approach for domain adaptation of a dependency parser via self-training. We improve parsing accuracy for out-of-domain texts with a self-training approach that uses confidence-based methods to select additional training samples. We compare two confidence-based methods: The first method uses the parse score of the employed parser to measure the confidence into a parse tree. The second method calculates the score differences between the best tree and alternative trees. With these methods, we were able to improve the labeled accuracy score by 1.6 percentage points on texts from a chemical domain and by 0.6 on average on texts of three web domains. Our improvements on the chemical texts of 1.5% UAS is substantially higher than improvements reported in previous work of 0.5% UAS. For the three web domains, no positive results for self-training have been reported before.

1 Introduction

Semi-supervised techniques gain popularity since they allow the exploitation of unlabeled data and avoid the high costs for labeling new data, cf. (Sarkar, 2001; Steedman et al., 2003; McClosky et al., 2006a; Koo et al., 2008; Søgaard and Rishøj, 2010; Petrov and McDonald, 2012; Chen et al., 2013). For domain adaptation, semi-supervised techniques have been applied successfully, cf. (Reichart and Rappoport, 2007; Petrov and McDonald, 2012; Pekar et al., 2014). Self-training is one of these appealing techniques which improves parsing accuracy by using a parser's own annotations. In a self-training iteration, a base model is first trained on annotated corpora, the base model is then used to annotate unlabeled data, finally a self-trained model is trained on both manually and automatically annotated data. This procedure might be repeated several times.

Self-training has been successfully used for instance in constituency parsing for in-domain and out-of-domain parsing (McClosky et al., 2006a; McClosky et al., 2006b; Reichart and Rappoport, 2007; Sagae, 2010). McClosky et al. (2006a) used self-training for constituency parsing. In their approaches, self-training was most effective when the parser is retrained on the combination of the initial training set and the large unlabeled dataset generated by both the generative parser and the reranker. This leads to many subsequent applications on domain adaptation via self-training for constituency parsing (McClosky et al., 2006b; Reichart and Rappoport, 2007; Sagae, 2010; Petrov and McDonald, 2012), while for dependency parsing, self-training was only effective in few cases. The question why it does not work equally well for dependency parsing is still a question that has not been satisfactorily answered. The paper tries to shed some light on the question under which circumstances and why self-training is applicable. More precisely, this paper makes the following contributions:

1. We present an effective confidence-based self-training approach.

2. We compare two confidence-based methods to select training sentences for self-training.

3. We apply our approaches on three web domains as well as on a chemical domain and we successfully improved the parsing performances for all tested domains.

The remainder of this paper is organized as follows: In Section 2, we give an overview of related work. In Section 3, we introduce two approaches to self-training and apply those on parsing of out-of-domain data. In Section 4, we describe the data and the experimental set-up. In Section 5, we present and discuss the results. Section 6 presents our conclusions.

2 Related Work

Charniak (1997) applied self-training to PCFG parsing, but this first attempt to self-training for parsing failed.

Steedman et al. (2002) implemented self-training and evaluated it using several different settings. They parsed 30 sentences per iteration while the training data contained 10K sentences. Experiments with multiple iterations showed moderate improvements only which is caused probably by the small number of additional sentences used for self-training.

McClosky et al. (2006a) reported strong results with an improvement of 1.1 F-score using the Charniak-parser, cf. (Charniak and Johnson, 2005). McClosky et al. (2006b) applied the method later on out-of-domain texts which show good accuracy gains too.

Reichart and Rappoport (2007) showed that self-training can improve the performance of a constituency parser without a reranker when a small training set is used.

Sagae (2010) investigated the contribution of the reranker for a constituency parser. The results suggest that constituency parsers without a reranker can achieve significant improvements, but the results are still higher when a reranker is used.

In the SANCL 2012 shared task self-training was used by most of the constituency-based systems, cf. (Petrov and McDonald, 2012), which includes the top ranked system, this indicates that self-training is already an established technique to improve the accuracy of constituency parsing on out-of-domain data, cf. (Le Roux et al., 2012). However, none of the dependency-based systems used self-training in the SANCL 2012 shared task.

One of the few successful approaches to self-training for dependency parsing was introduced by Chen et al. (2008). Chen et al. (2008) improved the unlabeled attachment score about one percentage point for Chinese. Chen et al. (2008) added parsed sentences that have a high ratio of dependency edges that span only a short distance, i.e., the head and dependent are close together. The rationale for this procedure is the observation that short dependency edges show a higher accuracy than longer edges.

Kawahara and Uchimoto (2008) used a separately trained binary classifier to select reliable sentences as additional training data. Their approach improved the unlabeled accuracy of texts from a chemical domain by about 0.5%.

Goutam and Ambati (2011) applied a multi-iteration self-training approach on Hindi to improve parsing accuracy within the training domain. In each iteration, they add 1,000 additional sentences to a small initial training set of 2,972 sentences, the additional sentences were selected due to their parse scores. They improved upon the baseline by up to 0.7% and 0.4% for labeled and unlabeled attachment scores after 23 self-training iterations.

Plank (2011) applied self-training with single and multiple iterations for parsing of Dutch using the Alpino parser (Malouf and Noord, 2004), which was modified to produce dependency trees. She found self-training produces only a slight improvement in some cases but worsened when more unlabeled data was added.

Plank and Søgaard (2013) used self-training in conjunction with dependency triplets statistics and the similarity-based sentence selection for Italian out-of-domain parsing. They found that the effect of self-training is unstable and does not lead to an improvement.

Cerisara (2014) and Björkelund et al. (2014) applied self-training to dependency parsing on nine languages. Cerisara (2014) could only report negative results when they apply the self-training approach for dependency parsing. Similarly, Björkelund et al. (2014) could observe only on Swedish a positive effect.

For our approaches, confidence-based methods have been shown to be crucial such as by Dredze et al. (2008) and Crammer et al. (2009). These methods provide estimations on the quality of the predictions.

Mejer and Crammer (2012) used confidence-based methods to measure the prediction quality of a dependency parser. The confidence scores generated by these methods are correlated with the prediction accuracy of the dependency parser, i.e. higher confidence is correlated with high accuracy scores.

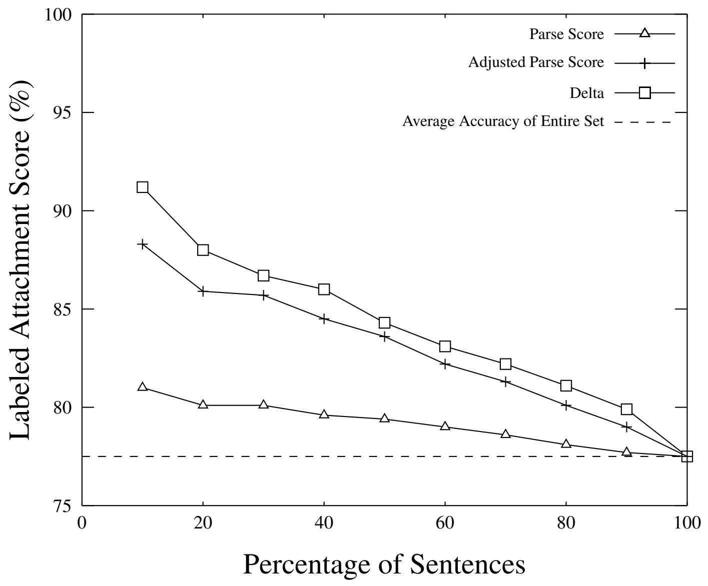


Figure 1: The graph shows the outcome of an experiment on the development set when the sentences were sorted due to the confidence score and the accuracy of the top n percent is computed. The y-axis shows the accuracy and the x-axis the percentage of the number of sentences that were considered. Each curve represents a selection method.

3 Self-training

The hypotheses for our experiments is that the selection of high-quality dependency trees is a crucial precondition for the successful application of self-training to dependency parsing. Therefore, we explore two confidence-based methods to select such dependency trees from newly parsed sentences. Our self-training approach consists of the following steps:

1. A parser is trained on the source domain training set in order to generate a base model.
2. We analyze a large number of unlabeled sentences from a target domain with the base model.
3. We build a new training set consisting of the source domain corpus and parsed sentences that have a high confidence score.
4. We retrain the parser on the new training set in order to produce a self-trained model.
5. Finally, we use the self-trained model to parse the target domain test set.

We test two methods to gain confidence scores for a dependency tree. The first method uses the parse scores, which is based on the observation that a higher parse score is correlated with a higher parsing quality. The second method uses the method of Mejer and Crammer (2012) to compute the *Delta* score. Mejer and Crammer (2012) compute a confidence score for each edge. The algorithm attaches each edge to an alternative head. The Delta is the score difference between the original dependency tree and the tree with the changed edge. This method provides a per-edge confidence score. Note that the scores are real numbers and might be greater than 1. We changed the Delta-approach in two aspects from that of Mejer and Crammer (2012). The new parse tree contains a node that has either a different head or might have a different edge label or both since we use labeled dependency trees in contrast to Mejer and Crammer (2012). To obtain a single score for a tree, we use the averaged score of all score differences gained for each edge by the 'Delta'-method.

We use the Mate tools[1] to implement our self-training approach. The Mate tools contain a part-of-speech (pos) tagger, morphological tagger, lemmatizer, graph-based parser and an arc-standard transition-based parser. The arc-standard transition-based parser has the option to use a graph-based model to rescore the beam. The

[1]https://code.google.com/p/mate-tools/

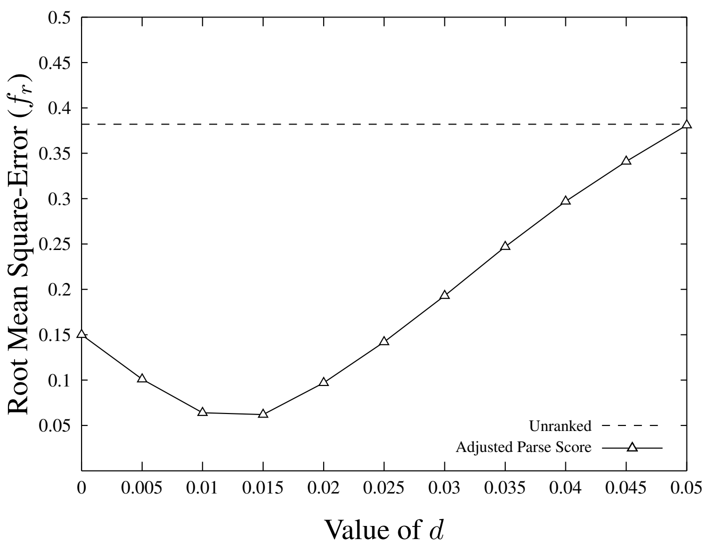


Figure 2: The root mean square-error (f_r) of development set after ranked by adjusted parse scores with different values of d.

parser has further the option to use a joint tagger and parser. The joint system is able to gain a higher accuracy for both part-of-speech tagging and parsing compared to a pipeline system.

We use the arc-standard transition-based parser which employs beam search and a graph-based rescoring model. This parser computes a score for each dependency tree by summing up the scores for each transition and dividing the score by the total number of transitions, due to the swap-operation (used for non-projective parsing), the number of transitions can vary, cf. (Kahane et al., 1998; Nivre, 2007).

Our second confidence-based method requires the computation of the score differences between the best tree and alternative trees. To compute the smallest difference (Delta), we modified the parser to derive the parse trees that contains the highest scoring alternative that replaces a given edge with an alternative one. This means either that the dependent is attached to another node or the edge label is changed, or both the dependent is attached to another node and the edge is relabeled. More precisely, during the parsing for alternative trees, beam candidates that contain the specified labeled edge will be removed from the beam at the end of each transition. Let $Score_{best}$ be the score of the best tree, $Score_i$ be the score of the alternative tree for the i_{th} labeled edge and L be the length of the sentence, the Delta ($Score_{Delta}$) for a parse tree is then calculated as follows:

$$Score_{Delta} = \frac{\sum_{i=1}^{L} |Score_{best} - Score_i|}{L} \qquad (1)$$

To obtain high-accuracy dependency trees is crucial for our self-training approaches, thus we first assess the performance of the confidence-based methods on the development set for the selection of high-quality dependency trees. We rank the parsed sentences by their confidence scores in a descending order. Figure 1 shows the accuracy scores when selecting 10-100% of sentences with an increment of 10%. The Delta method shows the best performance for detecting high-quality parse trees, we observed that when inspecting 10% of sentences, the accuracy score difference between the Delta method and the average score of the entire set is nearly 14%. The method using the parse score does not show such a high accuracy difference. The accuracy of the 10% top ranked sentences are lower.

We observed that despite that the parse score is the averaged value of a sequence of transitions of a parse, long sentences generally exhibit a higher score. Thus, short sentences tend to be ranked in the bottom, even if they might have a higher accuracy than longer sentences. To reduce the dependency of the score on the sentence length and to maximize the correlation of the score and the accuracy, we adjust the scores for each parse tree by

	train	**test**			**dev**
	CoNLL09	**Weblogs**	**Newsgroups**	**Reviews**	**Weblogs**
Sentences	39,279	2,141	1,195	1,906	2,150
Tokens	958,167	40,733	20,651	28,086	42,144

Table 1: The size of the source domain training set and target domain test datasets for web domain evaluation.

	Weblogs	**Newsgroups**	**Reviews**
Sentences	513,687	512,000	512,000
Tokens	9,882,352	9,373,212	7,622,891

Table 2: The size of unlabeled datasets for web domain evaluation.

subtracting from them a constant d multiplied by the sentence length (L). The new parse scores are calculated as follow:

$$Score_{adjusted} = Score_{original} - L \times d \quad (2)$$

To obtain the constant d, we apply the defined equation to all sentences of the development set and rank the sentences due to their adjusted scores in a descending order. The value of d is selected to minimize the root mean square-error (f_r) of the ranked sentences. Similar to Mejer and Crammer (2012) we compute the f_r by:

$$f_r = \sqrt{\sum_i n_i(c_i - a_i)^2 / (\sum_i n_i)} \quad (3)$$

We use 100 bins to divide the accuracy into ranges of one percent, parse scores in the range of $[\frac{(i-1)\times 3}{100}, \frac{i\times 3}{100}]$ are assigned to the i_{th} bin[2]. Let n_i be the number of sentences in ith bin, c_i is defined as estimated accuracy of the bin calculated by $\frac{i-0.5}{100}$ and a_i is the actual accuracy of the bin. We calculate f_r by iterating stepwise over d from 0 to 0.05 with an increment of 0.005. Figure 2 shows the f_r for the adjusted parse scores with different values of d. The lowest f_r is achieved when $d = 0.015$, this reduce the f_r from 0.15 to 0.06 when compare to the parse score method without adjustment ($d = 0$). In contrast to the $f_r = 0.06$ calculated when d is set to 0.015, the unranked sentences have a f_r of 0.38, which is six times larger than that of the adjusted one. The reduction on f_r achieved by our adjustment indicates that the adjusted parse scores have a higher correlation to the accuracy when compare to the ones without the adjustment.

[2]We observed that parse scores computed by the parser are positive numbers and generally in the range of [0,3].

	train	**test**	**unlabeled**
Sentences	18,577	195	256,000
Tokens	446,573	5,001	6,848,072

Table 3: The size of datasets for chemical domain evaluation.

Figure 1 shows the performance of the adjusted parse scores for finding high accuracy parse trees in relation to the original parse score and the Delta-based method. The adjusted parse score-based method performs significantly better than that of the original score with a performance similar to the Delta method. The method based on the parse scores is faster as we do not need to apply the parser to find alternatives for each edge of a dependency tree.

4 Experimental Set-up

We apply the approaches on three web domains and chemical texts. Section 4.1 describes the datasets that we use in our experiments. Section 4.2 explains the parser and Section 4.3 the evaluation methods.

4.1 Datasets

Web Domain. Our experiments are evaluated on three web domains provided by Ontonotes v5.0[3] and the SANCL 2012 datasets. We use these datasets since sufficient unlabeled datasets that are required for self-training are provided by the SANCL 2012 shared task. We use the last 20%

[3]https://catalog.ldc.upenn.edu/LDC2013T19

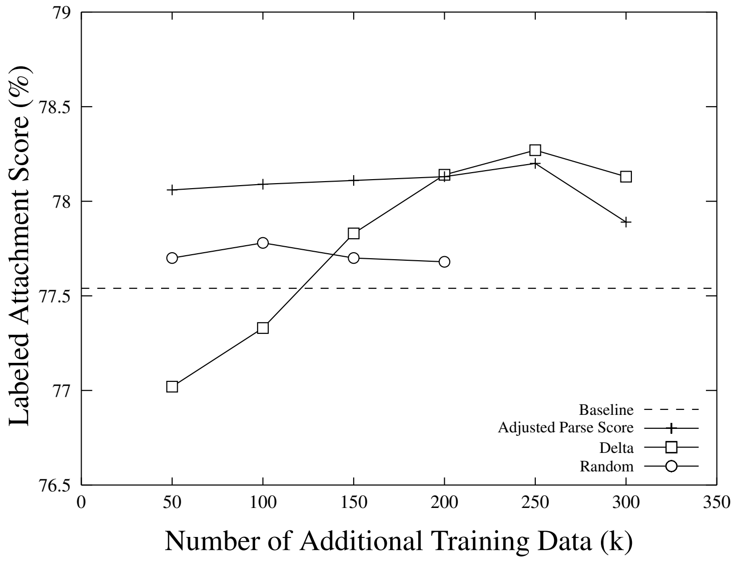


Figure 3: The effect of our self-training approaches on the weblogs development set.

of the weblogs section of the OntoNotes v5.0 corpus. Exact 50% of the selected sentences is used with SANCL Newsgroups and Reviews test data as test sets while the second half is used as a development set. We converted the datasets with the LTH constituent-to-dependency conversion tool, cf. (Johansson and Nugues, 2007). For the source domain training data, we use the CoNLL 2009 training dataset, cf. (Hajič et al., 2009). Table 1 shows the details for the training, development and test set. We use 500k of the SANCL unlabeled data for each domain after we pre-processed them by removing sentences that are longer than 500 tokens or containing non-English words which reduced the size of the datasets by 2%. Table 2 shows details about the amount of unlabeled texts.

Chemical Domain. To compare with previous work, we apply the approach on texts from the chemical domain that were prepared for the domain adaptation track of the CoNLL 2007 shared task, cf. (Nivre et al., 2007). Table 3 shows the details about the amount of available sentences for training, development and test set. The source data sets of the chemical domain are smaller than the ones for web domains. The training set has about half of the size. Thus we use only 250k unlabeled sentences from the chemical domain which share the same ratio of training set size to unlabeled data set size compared to the web domain data sets. To keep the same scale for training and unlabeled sets allows us easily adapt the best setting from web domain experiments.

4.2 Dependency Parser

We use the Mate transition-based dependency parser with default settings in our experiments, cf. Bohnet et al. (2013). For tagging, we use predicted pos tags to carry out the experiments as we believe that this is a more realistic scenario. The parser's internal tagger is used to supply the pos tags for both unlabeled sets and test datasets. In order to compare with previous work, we evaluate the approaches additionally on gold pos tags for texts of the chemical domain as gold tags were used by previous work.

The baselines are generated by training the parser on the source domain and testing the parser on the described target domain test sets.

4.3 Evaluation Method

For the evaluation of the parser's accuracy, we report labeled attachment scores (LAS). We included all punctuation marks in the evaluation.

For significance testing, we use the script provided by the CoNLL 2007 shared task which is Dan Bikel's randomized parsing evaluation comparator with the default settings of 10,000 iterations. The statistically significant results are marked due to their p-values, (*) p-value<0.05, (**) p-value<0.01.

5 Results and Discussion

Random Selection-based Self-training. As a baseline experiment, we apply self-training on

	PPOS		GPOS	
	LAS	UAS	LAS	UAS
Parse Score	80.8*	83.62*	83.44**	85.74**
Delta	81.1*	83.71*	83.58**	85.8**
Baseline	79.68	82.5	81.96	84.28
Kawahara (Self-trained)	-	-	-	84.12
Kawahara (Baseline)	-	-	-	83.58
Sagae (Co-training)	-	-	81.06	83.42

Table 5: The results of the adjusted parse score-based and the Delta-based self-training approaches on the chemical test set compared with the best-reported self-training gain (Kawahara and Uchimoto, 2008) and the best results of CoNLL 2007 shared task, cf. Sagae and Tsujii (2007). (PPOS: results based on predicted pos tags, GPOS: results based on gold pos tags, Self-trained: results of self-training experiments, Co-trained: results of co-training experiments.)

	PS	Delta	Baseline
Weblogs	79.80**	79.68**	78.99
Newsgroups	75.88**	75.87*	75.3
Reviews	75.43*	75.6**	75.07
Average	77.03	77.05	76.45

Table 4: The effect of the adjusted parse score-based (PS) and the Delta-based self-training approaches on weblogs, newsgroups and reviews test sets.

randomly selected sentences that we add to the training set. Figure 3 shows an overview of the results. We obtain an improvement of 0.24% which is not statistically significant. This finding is in line with related work when applying non-confidence-based self-training approaches to dependency parsing, cf. (Cerisara, 2014; Björkelund et al., 2014).

Parse Score-based Self-training. For the parse score-based method, we add between 50k to 300k parsed sentences from the weblogs dataset that have been sorted by their parse scores in descending order. Figure 3 illustrates that the accuracy increase when more parsed sentences are included into the training set, we obtain the largest improvement of 0.66% when we add 250k sentences, after that the accuracy starts to decrease.

Delta-based self-training. For our Delta-based approach, we select additional training data with the Delta method. We train the parser by adding between 50k to 300k sentences from the target domain. We gain the largest improvement when we add 250k sentences to the training set, which improves the baseline by 0.73% (cf. Figure 3). We observe that the accuracy starts to decrease when we add 50k to 100k sentences. Our error analysis shows that these parse trees are mainly short sentences consisting of only three words. These sentences contribute probably no additional information that the parser can exploit.

Evaluating on Test Sets. We adapt our best settings of 250k additional sentences for both approaches and apply them to the web test sets (weblogs, newsgroups and reviews). As illustrated in Table 4, all results produced by both approaches are statistically significant improvements compared to the baseline. Our approach achieves the largest improvement of 0.81% with the parse score-based method on weblogs. For the Delta-based method, we gain the largest improvement of 0.69% on weblogs. Both approaches achieve similar improvements on newsgroups (0.57% and 0.58% for Delta and parse score-based methods, respectively). The Delta method performs better on reviews with an improvement of 0.53% vs. 0.36%. Both approaches improve on average by 0.6% on the three web domains.

We further evaluate our best settings on chemical texts provided by the CoNLL 2007 shared task. We adapt the best settings of the web domains and apply both confidence-based approaches to the chemical domain. For the constant d, we use 0.015 and we use 125k additional training data out of the 250k from the unlabeled data of the chemical domain. We evaluate our confidence-based methods on both predicted and gold pos tags. After retraining, both confidence-based methods achieve significant improvements in all experiments. Table 5 shows the results for the texts of the chemical domain. When we use predicted pos tags, the Delta-based method gains an improvement of

1.42% while the parse score-based approach gains 1.12%. For the experiments based on gold tags, we achieve a larger improvements of 1.62% for the Delta-based and 1.48% for the parse score-based methods.

Table 5 compares our results with that of Kawahara and Uchimoto (2008). We added also the results of Sagae and Tsujii (2007) but those are not directly comparable since they were gained with co-training. Sagae and Tsujii (2007) gained additional training data by parsing the unlabeled data with two parsers and then they select those sentence where the parsers agree.

Kawahara and Uchimoto (2008) reported positive results for self-training. They use a separate trained binary classifier to select additional training data. Kawahara and Uchimoto (2008) did evaluations only on gold pos tags. Our baseline is higher than Kawahara and Uchimoto (2008)'s self-training result, starting from this strong baseline, we could improve by 1.62% LAS and 1.52% UAS which is an error reduction of 9.6% on the UAS (cf. Table 5). The largest improvement of 1.52% compared to that of Kawahara and Uchimoto (2008) (0.54% UAS) is substantially larger. We obtained the result by a simple method and we do not need a separately trained classifier.

The confidence scores have shown to be crucial for the successful application of self-training for dependency parsing. In contrast to constituency parsing, self-training for dependency parsing does not work or at least not well without this additional confidence-based selection step. The question about a possible reason for the different behavior of self-training in dependency parsing and in constituency parsing remains open and only speculative answers could be given. We plan to investigate this further in the future.

6 Conclusions

In this paper, we introduced two novel confidence-based self-training approaches to domain adaptation for dependency parsing. We compared a self-training approach that uses random selection and two confidence-based approaches. While the random selection-based self-training method did *not* improve the accuracy which is in line with previously published negative results, the two confidence-based methods were able to gain statistically significant improvements and show a relative high accuracy gain.

The two confidence-based approaches achieve statistically significant improvements on all four test domains which are weblogs, newsgroups, reviews and the chemical domain. In the web domains, we gain up to 0.8 percentage points and on average both approaches improve the accuracy by 0.6%. In the chemical domain, the Delta-based and the parse score-based approaches gain 1.42% and 1.12% respectively when using predicted pos tags. When we use gold pos tags, both approaches achieved a larger improvement of 1.62% with the Delta method and 1.48% with the parse score method. In total, our approaches achieve significantly better accuracy for all four domains.

We conclude from the experiments that self-training based on confidence is worth applying in a domain adaptation scenario and that a confidence-based self-training approach seems to be crucial for the successful application of self-training in dependency parsing. This paper underlines the finding that the preselection of parse trees is probably a precondition that self-training becomes effective in the case of dependency parsing and to reach a significant accuracy gain.

Acknowledgments

We would like to thank John Barnden for discussions and comments as well as the anonymous reviewers for their helpful reviews.

References

Anders Björkelund, Özlem Çetinoğlu, Agnieszka Faleńska, Richárd Farkas, Thomas Mueller, Wolfgang Seeker, and Zsolt Szántó. 2014. The IMS-Wrocław-Szeged-CIS entry at the SPMRL 2014 Shared Task: Reranking and Morphosyntax meet Unlabeled Data. In *Proc. of the Shared Task on Statistical Parsing of Morphologically Rich Languages*.

Bernd Bohnet, Joakim Nivre, Igor Boguslavsky, Richard Farkas, Filip Ginter, and Jan Hajia. 2013. Joint morphological and syntactic analysis for richly inflected languages. *Transactions of the Association for Computational Linguistics*, 1:415–428.

Christophe Cerisara. 2014. Semi-supervised experiments at LORIA for the SPMRL 2014 Shared Task. In *Proc. of the Shared Task on Statistical Parsing of Morphologically Rich Languages*, Dublin, Ireland, August.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pages 173–180, Stroudsburg, PA, USA. Association for Computational Linguistics.

Eugene Charniak. 1997. Statistical parsing with a context-free grammar and word statistics. *AAAI/IAAI*, 2005:598–603.

Wenliang Chen, Youzheng Wu, and Hitoshi Isahara. 2008. Learning reliable information for dependency parsing adaptation. In *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*, pages 113–120. Association for Computational Linguistics.

Wenliang Chen, Min Zhang, and Yue Zhang. 2013. Semi-supervised feature transformation for dependency parsing. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1303–1313. Association for Computational Linguistics.

Koby Crammer, Alex Kulesza, and Mark Dredze. 2009. Adaptive regularization of weight vectors. In *Advances in Neural Information Processing Systems*, pages 414–422.

Mark Dredze, Koby Crammer, and Fernando Pereira. 2008. Confidence-weighted linear classification. In *Proceedings of the 25th international conference on Machine learning*, pages 264–271. ACM.

Rahul Goutam and Bharat Ram Ambati. 2011. Exploring self training for hindi dependency parsing. In *Proceedings of the 5th International Joint Conference on Natural Language Processing*, volume 2, pages 22–69.

Jan Hajič, Massimiliano Ciaramita, Richard Johansson, Daisuke Kawahara, Maria Antònia Martí, Lluís Màrquez, Adam Meyers, Joakim Nivre, Sebastian Padó, Jan Štěpánek, Pavel Straňák, Mihai Surdeanu, Nianwen Xue, and Yi Zhang. 2009. The conll-2009 shared task: Syntactic and semantic dependencies in multiple languages. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL): Shared Task*, pages 1–18.

Richard Johansson and Pierre Nugues. 2007. Extended constituent-to-dependency conversion for english. In *16th Nordic Conference of Computational Linguistics*, pages 105–112. University of Tartu.

Sylvain Kahane, Alexis Nasr, and Owen Rambow. 1998. Pseudo-projectivity: A polynomially parsable non-projective dependency grammar. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics (ACL) and the 17th International Conference on Computational Linguistics (COLING)*, pages 646–652.

Daisuke Kawahara and Kiyotaka Uchimoto. 2008. Learning reliability of parses for domain adaptation of dependency parsing. In *IJCNLP*, volume 8.

Terry Koo, Xavier Carreras, and Michael Collins. 2008. Simple semi-supervised dependency parsing. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 595–603.

Joseph Le Roux, Jennifer Foster, Joachim Wagner, Rasul Samad Zadeh Kaljahi, and Anton Bryl. 2012. Dcu-paris13 systems for the sancl 2012 shared task.

Robert Malouf and Gertjan Noord. 2004. Wide coverage parsing with stochastic attribute value grammars. In *In Proc. of IJCNLP-04 Workshop Beyond Shallow Analyses*.

David McClosky, Eugene Charniak, and Mark Johnson. 2006a. Effective self-training for parsing. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 152–159.

David McClosky, Eugene Charniak, and Mark Johnson. 2006b. Reranking and self-training for parser adaptation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 337–344. Association for Computational Linguistics.

Avihai Mejer and Koby Crammer. 2012. Are you sure?: Confidence in prediction of dependency tree edges. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, NAACL HLT '12, pages 573–576, Stroudsburg, PA, USA. Association for Computational Linguistics.

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 the CoNLL Shared Task of EMNLP-CoNLL 2007*, pages 915–932.

Joakim Nivre. 2007. Incremental non-projective dependency parsing. In *Proceedings of Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL HLT)*, pages 396–403.

Viktor Pekar, Juntao Yu, Mohab El-karef, and Bernd Bohnet. 2014. Exploring options for fast domain adaptation of dependency parsers. In *Proceedings of the First Joint Workshop on Statistical Parsing of Morphologically Rich Languages and Syntactic Analysis of Non-Canonical Languages*, pages 54–65, Dublin, Ireland, August. Dublin City University.

Slav Petrov and Ryan McDonald. 2012. Overview of the 2012 shared task on parsing the web. In *Notes of the First Workshop on Syntactic Analysis of Non-Canonical Language (SANCL)*, volume 59.

Barbara Plank and Anders Søgaard. 2013. Experiments in newswire-to-law adaptation of graph-based dependency parsers. In *Evaluation of Natural Language and Speech Tools for Italian*, pages 70–76. Springer Berlin Heidelberg.

Barbara Plank. 2011. *Domain Adaptation for Parsing*. Ph.D. thesis, University of Groningen.

Roi Reichart and Ari Rappoport. 2007. Self-training for enhancement and domain adaptation of statistical parsers trained on small datasets. In *ACL*, volume 7, pages 616–623.

Kenji Sagae and Jun'ichi Tsujii. 2007. Dependency parsing and domain adaptation with LR models and parser ensembles. In *Proceedings of the CoNLL Shared Task of EMNLP-CoNLL 2007*, pages 1044–1050.

Kenji Sagae. 2010. Self-training without reranking for parser domain adaptation and its impact on semantic role labeling. In *Proceedings of the 2010 Workshop on Domain Adaptation for Natural Language Processing*, pages 37–44. Association for Computational Linguistics.

Anoop Sarkar. 2001. Applying co-training methods to statistical parsing. In *Proceedings of the Second Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 175–182.

Anders Søgaard and Christian Rishøj. 2010. Semi-supervised dependency parsing using generalized tri-training. In *Proceedings of the 23rd International Conference on Computational Linguistics*, COLING '10, pages 1065–1073, Stroudsburg, PA, USA. Association for Computational Linguistics.

Mark Steedman, Steven Baker, Jeremiah Crim, Stephen Clark, Julia Hockenmaier, Rebecca Hwa, Miles Osborne, Paul Ruhlen, and Anoop Sarkar. 2002. Semi-supervised training for statistical parsing.

Mark Steedman, Rebecca Hwa, Miles Osborne, and Anoop Sarkar. 2003. Corrected co-training for statistical parsers. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 95–102.

Combining Active Learning and Partial Annotation for Domain Adaptation of a Japanese Dependency Parser

Daniel Flannery[*]
Vitei Inc.
Kankoboko-cho 79 Shimogyo-ku,
Kyoto, Japan
danielflannery@gmail.com

Shinsuke Mori
ACCMS, Kyoto University
Yoshida Honmachi, Sakyo-ku,
Kyoto, Japan
forest@i.kyoto-u.ac.jp

Abstract

The machine learning-based approaches that dominate natural language processing research require massive amounts of labeled training data. Active learning has the potential to substantially reduce the human effort needed to prepare this data by allowing annotators to focus on only the most informative training examples. This paper shows that active learning can be used for domain adaptation of dependency parsers, not just in single-domain settings. We also show that entropy-based query selection strategies can be combined with partial annotation to annotate informative examples in the new domain without annotating full sentences. Simulations are common in work on active learning, but we measured the actual time needed for manual annotation of data to better frame the results obtained in our simulations. We evaluate query strategies based on both full and partial annotation in several domains, and find that they reduce the amount of in-domain training data needed for domain adaptation by up to 75% compared to random selection. We found that partial annotation delivers better in-domain performance for the same amount of human effort than full annotation.

1 Introduction

Active learning is a promising approach for domain adaptation because it offers a way to reduce the amount of data needed to train classifiers, minimizing the amount of difficult in-domain annotation. This type of annotation requires annotators to have both domain knowledge plus familiarity with annotation standards. There has been much recent work on active learning for a variety of natural language processing tasks (Olsson, 2009), but most of it is concerned only with the single-domain case. Additionally, work on active learning commonly reports results for simulations only because of the high cost of annotation work.

We use active learning to perform domain adaptation for a Japanese dependency parsing task, and measure the actual time required for manual annotation of training data to better frame the results of our experiments. This kind of evaluation is crucial for assessing the effectiveness of active learning in practice.

Previous work on active learning for structured prediction tasks like parsing (Hwa, 2004) often assumes that the training data must be fully annotated. But recent work on dependency parsing (Spreyer et al., 2010; Flannery et al., 2011) has shown that models trained from partially annotated data (where only part of the tree structure is annotated) can achieve competitive performance. However, deciding which portion of the tree structure to annotate remains a difficult problem.

2 Related Work

Most previous work on active learning for parsing (Hwa, 2004; Sassano and Kurohashi, 2010) studies the single-domain case, where the initial labeled data set and the pool of unlabeled data share the same domain. An important difference from previous work is that we focus on domain adaptation, so we assume that the initial labeled data and annotation pool come from different domains.

Previous work on active learning for parsing (Tang et al., 2002; Hwa, 2004) has focused on selecting sentences to be fully annotated. Sassano and Kurohashi (2010) showed that smaller units like phrases (*bunsetsu*) could also be used in an active learning scenario for a Japanese dependency parser. Their work included results for partially

[*]This work was done when the first author was at Kyoto University.

	i	1	2	3	4	5	6	7	8	9
annotation	w_i	政府	は	投資	に	つなが	る	と	歓迎	し
	Eng.	Gov.	*subj.*	investment	to	leads	*ending*	that	welcomes	do
	t_i	noun	part.	noun	part.	verb	infl.	part.	noun	verb
	d_i full	2	8	4	5	6	7	8	9	NULL
	d_i part.		8							
features	F1		6							
	F2		は						歓迎	
	F3		part.						noun	
	F4		NULL, NULL, 政府						つなが, る, と	
	F5		投資, に, つなが						し, NULL, NULL	
	F6		NULL, NULL, noun						verb, infl., part.	
	F7		noun, part., verb						verb, NULL, NULL	

The second word, the case marker は (*subj.*), has two grammatically possible heads: the verbs つなが (leads) and 歓迎 (welcomes). In the partial annotation framework, only this word needs to be annotated.

Table 1: An example of full annotation (d_i full) and partial annotation (d_i part.) for a sentence. Features for the dependency between the case marker は (*subj.*) and the verb 歓迎 (welcomes) are also shown.

annotated sentences, but did not use entropy-based query strategies (Tang et al., 2002; Hwa, 2004) designed for selecting whole sentences because of the difficulty of applying them. We use an even smaller unit, words, and show how entropy-based measures can be successfully applied to their selection.

Mirroshandel and Nasr (2011) also investigated selection of units smaller than sentences for a graph-based parser in the single-domain setting. Their query strategy used an entropy-based measure calculated from n-best lists, which are computationally expensive and require modification of the parser's edge scoring function to produce. In contrast, our query strategy is a simpler one that does not require n-best lists.

All of the work discussed here reports results for simulations only. This is common practice in active learning research because large-scale annotation is prohibitively expensive. Some recent work on active learning has started to include more realistic measures of the actual costs of annotation (Settles et al., 2008). In this paper, we measure the time needed to manually annotate sentences with dependencies to better understand the costs of active learning for dependency parsing.

3 MST Parsing

Currently, the two major types of data-driven dependency parsers are shift-reduce parsers (Nivre et al., 2006) and graph-based parsers (McDonald et al., 2005). Shift-reduce parsers perform parsing deterministically (so their time complexity can be as fast as linear in the size of the input). Graph-based dependency parsers treat parsing as the search for a directed maximum spanning tree (MST). We adopt the latter type in this paper because its accuracy is slightly higher especially for long sentences (McDonald and Nivre, 2011).

3.1 Partial Annotation

Our goal is to reduce the total cost of preparing data for domain adaptation. We do this by combining partial annotation with active learning. *Partial annotation* refers to an annotation method where only some dependencies in a sentence are annotated with their heads. The standard method in which all words must be annotated with heads is called *full annotation*. Table 1 shows an example of both types of annotation for a sentence.

Full sentences are the default unit of annotation in full annotation, even though the parser is trained from and operates on smaller units such as words or chunks. The motivation for partial annotation is to match the unit of annotation with the smallest unit that the parser uses for training. In the extreme case this is as small as a single dependency between two words. This fine-grained annotation unit is a natural fit for active learning, where we seek to find training examples with the greatest training value. However, fine-grained units are cognitively more difficult for a human annotator because less context is available. Thus, we must balance the granularity of annotations against the difficulty of processing them.

3.2 Pointwise MST Parsing

To enable the use of partial annotation in active learning, we use a pointwise MST parser (Flannery et al., 2011) that predicts each word's head independently. It uses only simple features based on surface forms and part-of-speech (POS) tags of words, and first-order features between pairs of head and dependent words. Higher-order features that refer to chains of two or more dependencies, like the ones used in the second-order MST introduced by McDonald and Pereira (2006), are not used. These restrictions make it easier to train on partially annotated sentences without sacrificing accuracy. Flannery et al. (2011) reported that both this parser and McDonald and Pereira (2006)'s second-order MST parser achieved just under 97% accuracy on a Japanese dependency parsing task. The assumption that written Japanese is head-final and that dependencies only go from left to right may be one reason why there is less of a performance gap between these two approaches than in other languages. They also reported that the training time of their parser is fifteen times faster than the second-order MST parser, making it easier to use with active learning.

The following features, both individually and as combination features, are used in the pointwise parser that we adopt.

F1: The distance $j - i$ between a dependent word w_i and its candidate head w_j.

F2: The surface forms w_i and w_j.

F3: The parts-of-speech of w_i and w_j.

F4: The surface forms of up to three words to the left of w_i and w_j.

F5: The surface forms of up to three words to the right of w_i and w_j.

F6: The parts-of-speech of the words selected for F4.

F7: The parts-of-speech of the words selected for F5.

Table 1 shows the values of these features for a partially annotated example sentence where one word, the case marker は (*subj.*), has been annotated with its head, the verb 歓迎 (welcomes). Partial annotation allows annotators to ignore trivial dependencies that are assumed to have little training value.

4 Partial Annotation as a Query Strategy

In this section we give some background on active learning and outline the query strategies that we use to identify informative training examples.

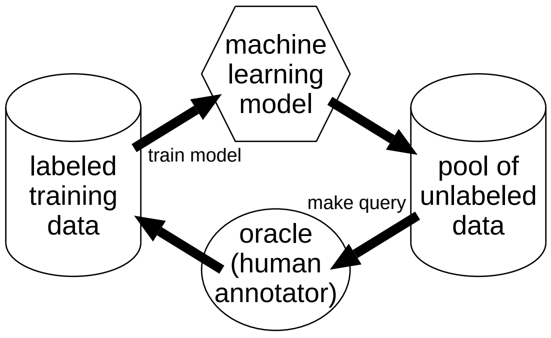


Figure 1: The pool-based active learning cycle.

4.1 Pool-Based Active Learning

We use the pool-based approach to active learning (Lewis and Gale, 1994), because it is a natural fit for domain adaptation. In this framework, we have both initial training data D_L (corresponding to labeled source domain corpora) and a large pool of unlabeled data D_U (corresponding to unlabeled target domain text) from which to choose training examples. While labeling domain-specific text is difficult, it is usually relatively easy to acquire (for example, from the web).

In each iteration the entire pool is evaluated sequentially and its members are ranked by their estimated training value as determined by some criterion, called the query strategy. The top instances are typically selected greedily. The basic flow of pool-based active learning is Figure 1 and described below.

1. Use a base learner B to train a classifier C from the labeled training set D_L.

2. Apply C to the unlabeled data set D_U and select I, the n most informative training examples.

3. Make a query to the oracle for the correct labels of training instances in I.

4. Move training instances in I from D_U to D_L.

5. Train a new classifier C' by applying B to D_L.

6. Repeat steps 2 to 5 until some stopping condition is fulfilled.

The stopping condition for terminating active learning depends on the application. It may be convenient to stop after a classifier C' with a given level of accuracy is reached or a fixed amount of data has been labeled. In a realistic domain adaptation scenario we are usually concerned with achieving a reasonable level of in-domain performance while keeping down annotation costs, so these kinds of simple termination criteria are sufficient.

4.2 Tree Entropy

Hwa (2004) proposed an active learning query strategy called tree entropy for selecting sentences to be fully annotated. Choosing a parse tree v for a sentence from the set of possible parse trees $\mathcal{V}$ is treated as assigning a value to the random variable V. The entropy of V,

$$H(V) = -\sum_{v \in \mathcal{V}} p(v) \log_2(p(v)), \qquad (1)$$

is equivalent to the expected number of bits needed to encode the distribution of possible parse trees. Here, $p(v)$ is the probability of assigning a single parse tree $V = v$ using a given parsing model. Distributions close to uniform have higher entropy, corresponding to higher uncertainty of the model. Longer sentences will have more parse trees in $\mathcal{V}$ and thus a larger value of $H(V)$. To compare sentences of varying lengths we normalize $H(V)$ by the log of the number of parse trees in $\mathcal{V}$.

$$H_n(V) = \frac{H(V)}{\log_2(|\mathcal{V}|)} \qquad (2)$$

4.3 1-Stage Selection

To use tree entropy as a strategy for partial annotation, we propose to change the unit of selection to words as follows. Consider a word w_i in an input sentence $\boldsymbol{w} = \langle w_1, w_2, \dots, w_n \rangle$, tagged with POS tags $\boldsymbol{t} = \langle t_1, t_2, \dots, t_n \rangle$ by a tagger. We will model the distribution of its possible heads, or head entropy. Let w_j be a single head word for w_i, where $j > i$ and $w_j \neq w_i$[1]. Then we can redefine v as a choice of position j and $\mathcal{V}$ as the set of legal values for j. Thus $p(v)$ becomes the probability of choosing the word at position j as the head of the one at position i. The parser we use (Flannery et al., 2011) calculates $p(v) = p(j|i)$ as follows. The feature vector $\boldsymbol{\phi} = \langle \phi_1, \phi_2, \dots, \phi_m \rangle$ consists of real values calculated from features on pairs (i, j) along with their contexts $\boldsymbol{w}$ and $\boldsymbol{t}$, with corresponding weights given by the parameter vector $\boldsymbol{\theta} = \langle \theta_1, \theta_2, \dots, \theta_m \rangle$.

$$p(j|\boldsymbol{w}, \boldsymbol{t}, i, \boldsymbol{\theta}) = \frac{\exp(\boldsymbol{\theta} \cdot \boldsymbol{\phi}(i,j))}{\sum_{j' \in \mathcal{J}} \exp(\boldsymbol{\theta} \cdot \boldsymbol{\phi}(i,j'))} \qquad (3)$$

The simplest way to combine this query strategy with partial annotation is to calculate the head entropy for each word appearing in a sentence in the annotation pool, and then choose individual words with the highest head entropy for annotation. We call this query strategy **1-stage**.

4.4 2-Stage Selection

We expect 1-stage to perform well at identifying words with high training value. However, in reality it is difficult to annotate heads for individual words without considering the overall sentence structure, so annotators must consider other dependencies. 1-stage does not realistically model annotation costs.

To address this problem, we propose a novel strategy called 2-stage which more accurately reflects the annotation process. It balances the ability to select fine-grained units for annotation against the difficultly of annotating them.

Words to annotate with heads are chosen in two steps. First, the entropy of each sentence in the pool is calculated by summing the head entropy of its words, and sentences are ranked from highest to lowest summed head entropy. Next, the sentence with the highest head entropy is chosen and the words it contains are ranked in decreasing order by their head entropy. A fixed proportion r of the highest-entropy words are then annotated. This value balances annotation granularity against annotation difficulty. A value of $r = 1.0$ is the standard full annotation case where all words are annotated with heads, which we refer to as **2-stage full**. A value of $r = 0.33$ means that the top 33% of the highest-entropy words in the sentence will be annotated, so we call this strategy **2-stage partial**[2]. In Section 5, we report results for these two values, though several were tried.

5 Evaluation

To evaluate the query strategies, we measured the reduction in target domain annotations needed to

[1]We assume that Japanese is a head-final language, and that each head w_j is located to the right of its dependent w_i in the sentence.

[2]We chose this value because it had good results in preliminary experiments where we tried values in the range from 0.1 to 1.0.

	ID	source	sentences	words	avg. length	dependencies
	EHJ-train	Dictionary examples	11,700	147,964	12.6	136,264
pool	NKN-train	Newspaper articles	9,023	263,425	29.2	254,402
	JNL-train	Journal abstracts	322	12,263	38.1	11,941
	NPT-train	NTCIR patents	450	18,378	40.8	17,928
test	NKN-test	Newspaper articles	1,002	29,037	29.0	28,035
	JNL-test	Journal abstracts	32	1,116	34.9	1,084
	NPT-test	NTCIR patents	50	2,275	45.5	2,225

Table 2: Sizes of corpora.

reach a certain level of in-domain accuracy. For the 2-stage strategy, we also measured how many dependencies a real annotator could annotate in a given time using partial and full annotation. Measuring the actual annotation time is important because our goal of active learning is to reduce the amount of human effort needed to prepare labeled training data for domain adaptation.

We used a corpus of example Japanese sentences from a dictionary as source domain training data (Mori et al., 2014). This data was used as to train the initial model in each experiment. We also collected Japanese text from three target domains: newspapers[3], journal article abstracts, and patents (Goto et al., 2011). For each domain, there is a large annotation pool of potential training examples and a smaller test set. See Table 2 for the details. Domain adaptation is needed in each case, because sentence length and vocabulary differs for each. Words in each sentence were manually segmented and assigned POS automatically with the tagger KyTea. This step can be done automatically because KyTea's F-measure score for word segmentation and POS tagging is about 98% (Neubig et al., 2011). Words were then manually annotated with their heads.

5.1 Number of Annotations

We first investigate how much strategies reduce the *number* of in-domain dependencies needed for domain adaptation. Because real annotation is costly and not strictly necessary to measure this reduction, we simulate active learning by selecting the gold standard dependency labels from the annotation pool. In practice, we are also concerned with the *time* needed for a human to annotate dependencies, which we examine in Section 5.3. Thus, good performance in this first experiment is

[3]The newspaper is similar to the Wall Street Journal and focuses on economics.

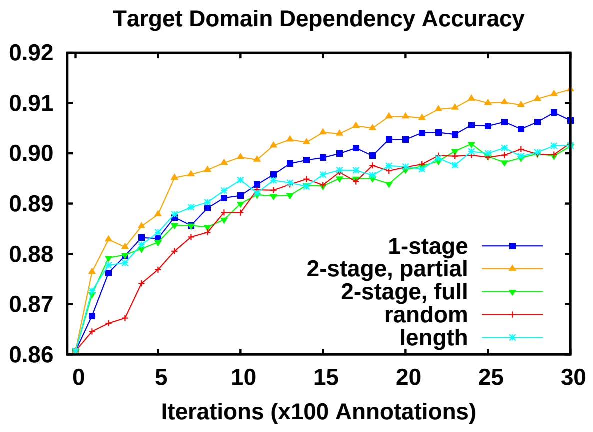


Figure 2: Newspaper (NKN) domain learning curves.

a necessary but not sufficient condition for an effective strategy. Because we assume that Japanese is a head-final language and heads always occur to the right of their dependents, for all strategies the last word in each sentence skipped. For 1-stage and 2-stage, we also skipped the second-to-last word in each sentence.

In addition to the 1-stage and 2-stage methods, we also tested two simple baselines. The strategy **random** simply selects words randomly from the pool. The **length** strategy simply chooses words with the longest possible dependency length[4]. This strategy reflects our intuition that long-distance dependencies are more difficult and thus more informative.

We use the dictionary example sentences (see Table 2) as the initial training set and performed thirty iterations of active learning. In each iteration, we select a batch of one hundred target domain dependency annotations, retrain the model, and then measure its in-domain accuracy.

[4]This is the same as selecting dependencies with the largest number of potential heads because we do not refer to the gold dependencies until after words have been selected.

Figure 2 shows the results for the newspaper domain. The accuracy of the random strategy increases slowly and peaks at just over 90.5%. For the first ten iterations the length strategy delivers an improvement over random, but performs essentially the same after that. This is probably because newspaper sentences are on average longer than dictionary examples (see Table 2), so at first words with the potential for longer dependencies are slightly more informative. However, this strategy is focused only on the training data and does not reflect the continuous updates of the model, and it soon begins to falter.

The 2-stage partial strategy dominates all other methods, though 1-stage reaches the same level after thirty-five iterations. Its peak accuracy is slightly higher than 91%, and it outperforms the best accuracy achieved by random after just seventeen iterations. In contrast, 2-stage full performs consistently worse than the partial annotation version, with behavior similar to length. While the 1-stage strategy always outperforms the random one, it lags behind 2-stage partial.

5.2 Annotation Pool Size

From Table 2, we can see that the size of the annotation pool for the newspaper domain is ten to twenty times as large as the ones for the other domains. The total number of dependencies selected is 3k, which is only 1.2% of the newspaper pool. Because the 2-stage strategy chooses some dependencies with lower entropy over competing ones with higher entropy from other sentences in the pool, we expect its accuracy to suffer when a much larger fraction of the pool is selected.

To investigate this effect, we created a smaller pool from NKN-train that is closer in size to the ones from the other domains. We used the first 12,165 dependencies for this smaller pool. The results are shown in Figure 3. It can be seen that 2-stage partial's lead over the 1-stage strategy has been eliminated. After seventeen rounds of annotation the 1-stage strategy begins to outperform the 2-stage strategy. The 2-stage partial strategy still dominates the 2-stage full strategy. This confirms our intuition that the relative performance of strategies is influenced by the size of the annotation pool. In general we expect the number of informative dependencies to increase as the pool size increases. Comparing these results with the results for the newspaper domain in Figure 2, we see that

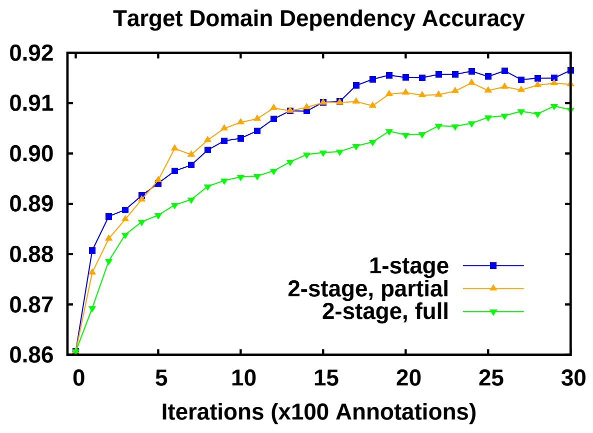


Figure 3: Newspaper (NKN) domain accuracy with a small annotation pool.

the 1-stage strategy is robust to changes in the pool size, but the 2-stage partial can outperform it for a very large pool.

5.3 Time Required for Annotation

Simulation experiments are still common when using active learning because the cost of annotation is very high. However, recently there is increased interest in measuring the true costs of annotation work when doing active learning (Settles et al., 2008). For a more realistic evaluation of active learning for parsing, we also measured annotation time for the 2-stage strategy. We trained a model on EHJ-train plus NKN-train and used this model and the 2-stage strategy to select dependencies to be annotated by a human annotator. The pool is 747 blog sentences[5] from the Balanced Corpus of Contemporary Written Japanese (Maekawa, 2008). We selected 2k dependencies in a single iteration so the annotator did not need to wait while the model was retrained after each batch of annotations. While real annotation times are not constant, this simplification is justified because we expect the annotation strategy (partial or full) to have a larger effect on the overall annotation speed than the dependencies that are selected.

A single annotator performed annotations for one hour each using the 2-stage strategy with both partial annotation and full annotation, alternating strategies every fifteen minutes. Sentences with more than forty words were not presented to the annotator. Table 3 shows the total number of dependencies annotated after each time period. After

[5]This data was taken from the Yahoo! Blog (OY) subcorpus.

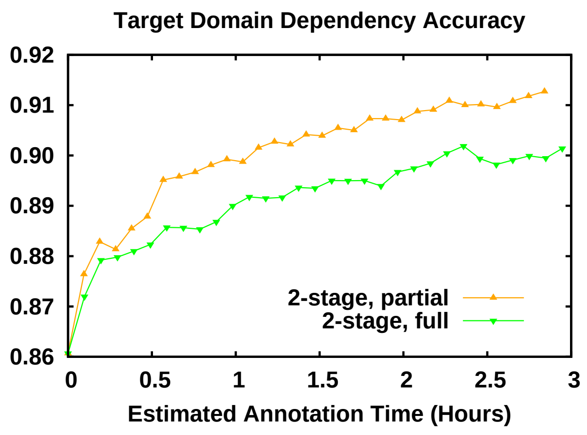


Figure 4: Estimated annotation time for the newspaper (NKN) domain.

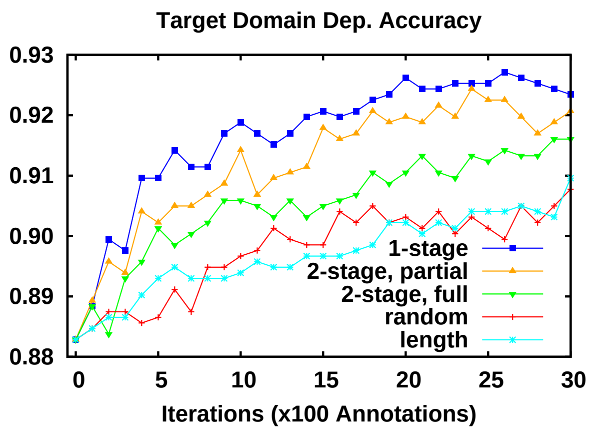


Figure 5: Journal (JNL) domain learning curves.

method	0.25 [h]	0.5 [h]	0.75 [h]	1.0 [h]
partial	226	458	710	1056
full	141	402	756	1018

Table 3: Annotation times for 2-stage methods.

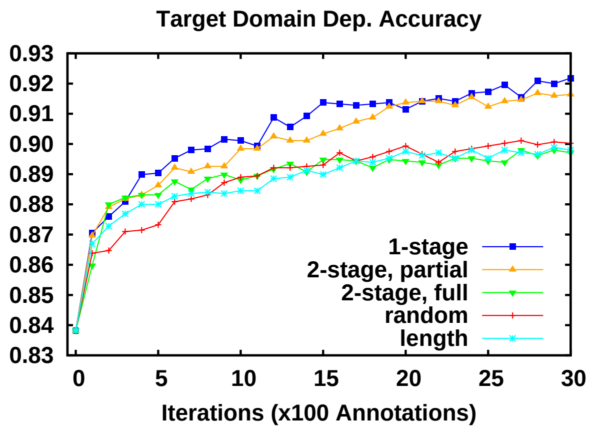


Figure 6: Patent (NPT) domain learning curves.

the first fifteen minutes, the annotator completed 226 annotations compared with 141 for full annotation, an increase of about 60%. However, as time progresses the difference becomes smaller, and after one hour the number of annotations was almost identical for both strategies.

From Table 3, we can see that the annotation speed reaches a maximum of about 350 annotations per fifteen minutes in the full annotation case, or 1.4k dependencies per hour. We expected more annotations to be completed when full annotation was used, because sentences have many trivial dependencies. However, the annotator reported that it was frustrating to check the annotation standard and how it handled subtle linguistic phenomena. Most of this work can be skipped when using partial annotation because the annotator was allowed to delete the estimated heads, so the annotation speeds ended up being almost identical. This result shows the importance of accurately modeling the annotation costs in active learning.

For both methods, the average speed is around 1k dependencies per hour. We used these speeds to estimate the rate of annotation for the experiments from Section 5.1. While this is not entirely realistic because speeds are likely to vary across domains, it is sufficient for comparing the relative performance of strategies in the same domain. The results are shown in Figure 4. We can see that accuracy improves faster for partial than it does for full, and the difference becomes pronounced after about half an hour of annotation work. In summary, partial annotation is more efficient and thus delivers a greater return on investment than full annotation for the proposed query strategy.

5.4 Results for Additional Domains

We also tested the proposed method in two additional domains. See Table 2 for the details of these corpora. Figure 5 and Figure 6 show results for the journal and patent domains, respectively. In these domains, 2-stage partial failed to outperform the 1-stage strategy. However, it still performed better than the 2-stage full strategy. As discussed in Section 5.2, the performance of the proposed method suffers when a large portion of the dependencies in the pool are selected. Here, the 3k dependencies that are selected are a much larger fraction of the pool – specifically, 16.7% for the patent domain and 25.1% for the journal domain. As in the

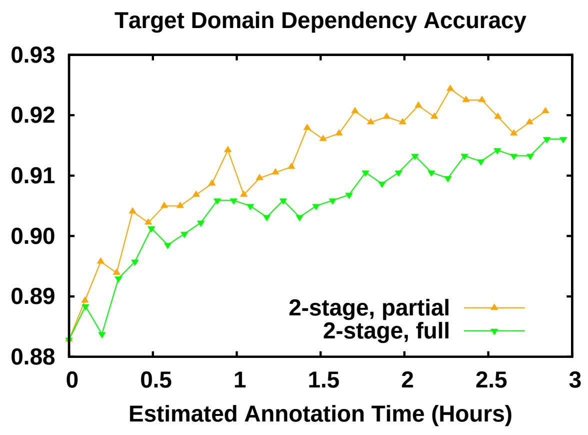


Figure 7: Estimated annotation time for the journal (JNL) domain.

domain	random	full	partial
NKN	3,000	–	1,300
JNL	3,000	1,800	900
NPT	2,700	–	1,500

Table 4: Reduction in in-domain data.

newspaper domain, in the patent domain the performance of 2-stage with full annotation is better than random for the first few iterations but soon becomes similar. This is not true in the journal domain, where this strategy consistently beats random. The length strategy edges out random for a few iterations in both domains, but ultimately their performance is similar.

Table 4 shows the number of annotations needed for the highest accuracy by the random baseline in the second column, while the next two show the number of annotations needed for the full and partial versions of 2-stage to outperform it. Thus, smaller numbers are better. Compared to the random strategy, 2-stage full had mixed results. In the journal abstract domain (JNL), it outperformed the random baseline while using only 60% of the amount of labeled data. However, it failed to outperform random selection in the other two domains. In contrast, 2-stage partial consistently outperforms random with only 45% to 55% of the labeled data. In terms of target domain data that must be prepared, it is clear that 2-stage partial offers large savings compared to random. It also does so more consistently and with less data than 2-stage full.

We also plotted the results for these domains in terms of estimated annotation time as we did

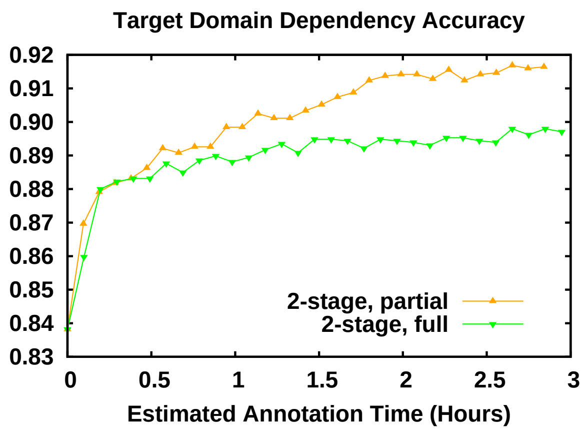


Figure 8: Estimated annotation time for the patent (NPT) domain.

in Section 5.3. Figure 7 shows the results for the journal domain and Figure 8 shows the results for the patent domain. As before, 2-stage full is more efficient than 2-stage partial. In these domains, partial dominates full after about one hour of annotation work. The gap between them is largest for the patent domain and smallest for the journal domain.

6 Conclusions

We combined partial annotation with active learning to adapt a Japanese dependency parser to new domains, and showed that active learning is not limited to single-domain settings. We showed that an entropy-based query strategy can successfully identify units smaller than sentences, and that partial annotation can be successfully applied to active learning of structured prediction tasks like parsing. This strategy reduced the amount of in-domain training data needed for domain adaptation by up to 75%. We also investigated how the overall size of the annotation pool affects the performance of the query strategy, and found that the proposed method performs best for large annotation pools.

To more accurately frame our results, we measured the actual annotation time required by a human annotator to prepare labeled data using different strategies. Using these results to estimate annotation times for earlier experiments, we showed that for the proposed method partial annotation is more efficient in terms of in-domain performance obtained per unit of annotation time than full annotation.

Acknowledgments

This work was supported by JSPS Grants-in-Aid for Scientific Research Grant Numbers 26280084 and 26540190, NTT agreement dated 06/19/2015, and Basic Research on Corpus Annotation project of The National Institute for Japanese Language and Linguistics. We are also grateful to the annotators for their contribution to the design of the annotation guidelines and their efforts in following them.

References

Daniel Flannery, Yusuke Miayo, Graham Neubig, and Shinsuke Mori. 2011. Training dependency parsers from partially annotated corpora. In *Proceedings of the Fifth International Joint Conference on Natural Language Processing*, Chiang Mai, Thailand.

Isao Goto, Bin Lu, Ka Po Chow, Eiichiro Sumita, and Benjamin K. Tsou. 2011. Overview of the patent machine translation task at the NTCIR-9 workshop. In *Proceedings of NTCIR-9 Workshop Meeting*, pages 559–578.

R. Hwa. 2004. Sample selection for statistical parsing. *Computational Linguistics*, 30(3).

D. Lewis and W. Gale. 1994. A sequential algorithm for training text classifiers. In *Proceedings of the 17th annual ACM SIGIR conference on research and development in information retrieval*.

Kikuo Maekawa. 2008. Balanced corpus of contemporary written Japanese. In *Proceedings of the 6th Workshop on Asian Language Resources*.

Ryan McDonald and Joakim Nivre. 2011. Analyzing and integrating dependency parsers. *Computational Linguistics*, 37(4):197–230.

Ryan McDonald and Fernando Pereira. 2006. Online Learning of Approximate Dependency Parsing Algorithms. In *Proceedings of the Eleventh European Chapter of the Association for Computational Linguistics*, volume 6.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 523–530.

Seyed Abolghasem Mirroshandel and Alexis Nasr. 2011. Active learning for dependency parsing using partially annotated sentences. In *Proceedings of the 12th International Conference on Parsing Technologies*, Dublin, Ireland.

Shinsuke Mori, Hideki Ogura, and Tetsuro Sasada. 2014. A Japanese word dependency corpus. In *Proceedings of the Nineth International Conference on Language Resources and Evaluation*, pages 753–758.

Graham Neubig, Yosuke Nakata, and Shinsuke Mori. 2011. Pointwise prediction for robust, adaptable Japanese morphological analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*.

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*.

F. Olsson. 2009. A literature survey of active machine learning in the context of natural language processing. Technical Report T2009:06, Swedish Institute of Computer Science.

Manabu Sassano and Sadao Kurohashi. 2010. Using smaller constituents rather than sentences in active learning for Japanese dependency parsing. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*.

B. Settles, M. Craven, and L. Friedland. 2008. Active learning with real annotation costs. In *Proceedings of the NIPS Workshop on Cost-Sensitive Learning*.

Kathrin Spreyer, Lilja Øvrelid, and Jonas Kuhn. 2010. Training parsers on partial trees: a cross-language comparison. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation*.

Min Tang, Xiaoqiang Luo, and Salim Roukos. 2002. Active learning for statistical natural language parsing. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*.

Incorporating Complementary Annotation to a CCGbank for Improving Derivations for Japanese

Sumire Uematsu and **Yusuke Miyao**
National Institute of Informatics / JST, PRESTO
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo, Japan
{uematsu,yusuke}@nii.ac.jp

Abstract

Wide-coverage resources for lexicalized grammars have been obtained by converting the existing treebanks into collections of derivations. Additional annotations to the source treebank can be used to improve these derivations. A treebank annotation called the NTT treebank was used for this paper to improve a CCGbank for Japanese. The source treebank of the CCGbank itself is created by automatically converting chunk-dependencies, but the CCGbank contains errors caused by noisier phrase structures and a lack of linguistic information, which is difficult to represent in chunk-dependency. The NTT treebank provides cleaner trees and functional and semantic information, e.g., coordinations and predicate-argument structures. The effect of the improvement process is empirically evaluated in terms of the changes in the dependency relations extracted from the resulting derivations.

1 Introduction

Wide-coverage resources for lexicalized grammars have been created by converting the existing treebanks into collections of derivations for the target grammars (Miyao and Tsujii, 2008; Hockenmaier and Steedman, 2007; Hockenmaier, 2006). However, the source corpora, such as the Penn Treebank (Marcus et al., 1993), often lack the necessary linguistic information for constructing these derivations, and this can create noise in the resulting derivations. Therefore, complementary annotations to the source treebank, e.g., NP bracketing and semantic roles, have been used used to improve the derivations (Honnibal et al., 2010; Vadas and Curran, 2008).

It is especially important to reduce the amount of noise in the derivations in the CCGbank for Japanese (Uematsu et al., 2015). Since the source treebank itself was created by converting the chunk-dependency, it potentially includes even more errors in the phrase structures and other types of information such as the functional tags. For instance, the chunk-dependency is often insufficient for correctly deciding on the phrase structure of coordinated NPs with modifiers. Since the dependencies do not encode the left boundary of each NP (Asahara, 2013), a manual annotation is needed for the precise structure. In fact, it essentially lacks any linguistic information which is difficult to represent in the chunk-dependency, e.g., the coordinated arguments.

The NTT treebank (Tanaka and Nagata, 2013) was used to improve the Japanese CCGbank for this paper. Basically the treebank is a manually corrected version of the source treebank used to create the CCGbank, but we treat the treebank as a collection of additional annotations to the source treebank. We specifically use its cleaner phrase structures to correct the phrase structure errors and its functional tags to properly deal with the coordinations in the derivations. Moreover, we use the annotations of causer roles in causative constructions in the NTT treebank, which are not available in the original syntactic resources, to recognize the arguments in the derivations. We show how the functional and semantic annotations on the treebank will be used for improving the CCGbank.

The improvement process together with the conversion in our previous work (Uematsu et al., 2015) can be regarded as a framework for obtaining a Japanese treebank and a derivation bank at a lower cost. That is, we can obtain a clean treebank containing rich linguistic information by 1) translating the existing syntactic resources, e.g., the chunk-dependency annotation, to a treebank, 2) manually correcting the phrase structures, and 3) using the cleaner treebank as a base for additional annotations. The treebank and the related

$$\begin{array}{lllll}
\mathrm{X/Y}: f & \mathrm{Y}: a & \rightarrow & \mathrm{X}: fa & (>) \\
\mathrm{Y}: a & \mathrm{X\backslash Y}: a & \rightarrow & \mathrm{X}: fa & (<) \\
\mathrm{X/Y}: f & \mathrm{Y/Z}: g & \rightarrow & \mathrm{X/Z}: \lambda x.f(gx) & (>\mathrm{B}) \\
\mathrm{Y\backslash Z}: g & \mathrm{X\backslash Y}: f & \rightarrow & \mathrm{X\backslash Z}: \lambda x.f(gx) & (<\mathrm{B})
\end{array}$$

Figure 1: Combinatory rules in Japanese CCG-bank.

resources are hopefully applicable to other grammar formalisms.

2 CCGbank for Japanese

2.1 Combinatory Categorial Grammar

Combinatory Categorial Grammar (CCG) is a lexicalized grammar formalism that is widely accepted in the NLP fields (Steedman, 2001). We briefly introduce its basic elements below.

A CCG grammar has two elements: *categories* for expressing the syntactic characteristic of the words and phrases and *combinatory rules* for combining the categories. There are two types of categories, *ground* and *complex*. *Ground categories* include S and NP, and *complex categories* are either X/Y or X\Y, where X and Y are the categories. Category X/Y means that it becomes a category X when it is combined with another category Y to its right, and X\Y means it takes on a category Y to its left. For example, categories S\NP and S/NP\NP represent an English intransitive verb and a transitive one, respectively.

Combinatory rules (Fig. 1) are applied to the categories to form categories for larger phrases. For example, a subject NP and intransitive verb S\NP are combined to form a sentence S by applying the backward application rule (< in Fig. 1). Figure 2 shows a CCG analysis of a simple Japanese sentence, which is called a *derivation*.

2.2 CCG-based syntactic theory for Japanese

We briefly describe Bekki's theoretical work on Japanese syntax (Bekki, 2010), which is the basis of the analysis in the Japanese CCGbank (Fig. 2). Based on CCG, his theory provides a comprehensive description for a variety of morphological and syntactic constructions, such as agglutination, scrambling, and long-distance dependencies.

There are three types of ground categories in his theory: S for sentences, NP for noun and postposition phrases, and CONJ for conjunctions. Categories S and NP have the syntactic features of *form* and *case*, respectively. Table 1 itemizes the values of these syntactic features.

Cat.	Feature	Value	Interpretation
NP	case	ga	nominative
		o	accusative
		ni	dative
		to	comitative, complementizer, etc.
		nc	none
S	form	stem	stem
		base	base
		neg	imperfect or negative
		cont	continuative
		vo_s	causative

Table 1: Features for Japanese syntax in (Uematsu et al., 2015).

Predicative words, such as verbs and adjectives, are represented as $\mathrm{S\backslash NP_{ga}}$, $\mathrm{S\backslash NP_{ni}\backslash NP_{ga}}$, etc., depending on their mandatory arguments. For example, $\mathrm{S\backslash NP_{ga}}$ is for intransitive verbs and for adjectives, and $\mathrm{S\backslash NP_{ga}\backslash NP_{o}}$ represents a transitive verb. Postpositions that work as argument markers include $\mathrm{NP_{ga}\backslash NP_{nc}}$ $\mathrm{NP_{ni}\backslash NP_{nc}}$, etc. For example, "が NOM ga" is represented as $\mathrm{NP_{ga}\backslash NP_{nc}}$ as it takes on the left NP to form a nominative NP. Postpositions can be used to form modifier phrases to verbal and adjective phrases. For example, "に ni" is S/S\NP if it takes on the left NP to form a temporal or a locative modifier.

The treatment of auxiliary verbs differs here from the English CCG. In Japanese, auxiliary verbs follow right after the main verb and express the semantic information, such as the tense and modality. For example, a verb "選ば/choose-NEG" and auxiliaries "なかっ/not-CONT" and "た/PAST-BASE" form a VP "選ばなかった", which means "did not choose". Auxiliary verbs are expressed as the category S\S, and the category is combined with a main verb via the function composition rule (<B in Fig. 1), as shown in Fig. 2.

Bekki's theory treats the coordination in a similar way as in (Steedman, 2001). There is a special rule for coordination Φ, with the restriction that X must be in a form of T/(T\$), e.g., $\mathrm{NP_{nc}/NP_{nc}}$ and S/NP\(S/NP).

$$\mathrm{X}_1 \quad \ldots \quad \mathrm{CONJ} \quad \mathrm{X}_m \rightarrow \quad \mathrm{X} \quad (\Phi) \qquad (1)$$

2.3 Japanese CCGbank

We proposed an algorithm in our previous work (Uematsu et al., 2015) to convert the existing chunk-dependency resources into CCG derivations for Japanese sentences. We refer to the collection of derivations obtained using this method as the original CCGbank for Japanese. Two steps

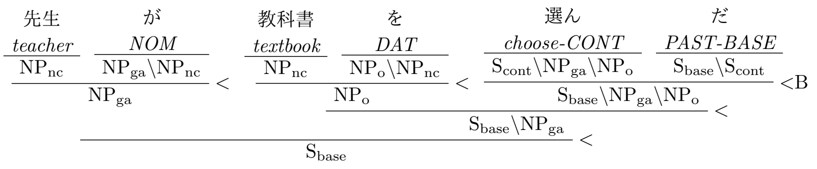


Figure 2: CCG derivation for Japanese sentence “The teacher chose the textbook.”

are needed to complete the conversion. First, we integrates chunk-dependencies from the Kyoto corpus (Kurohashi and Nagao, 2003) and a type of semantic role annotations from the NAIST corpus (Iida et al., 2007) to create tree structures with predicate argument structures (PASs). The trees are then translated into CCG derivations. 94% of the sentences in the source corpus were considered to be successfully converted into derivations. The lexicon extracted from the CCGbank has a lexical coverage of 99.4% and a sentential coverage of 87.0% on unseen text.

One of the obstacles in the conversion is often insufficient information for recovering the phrase structures. Several heuristic rules were used to complement for this lack, but we must make manual annotations, especially for the types of information that are difficult to represent in the chunk-dependency, e.g., coordinated arguments. The conversion errors due to the lack o information resulted in erroneous substructures in the trees and derivations and this leads to noises in the obtained grammar.

As a result, the grammar of the CCGbank is simplified for some constructions, specifically the coordinations. It treats the NP coordinations as noun-noun modifications. VP and ADJP coordinations are implicitly handled as a type of continuous clauses. By incorporating additional annotations into the derivation, our new procedure identifies the NP coordination and improves the substructure. On the other hand, we kept the VP coordinations as a type of continuous clauses, because it is difficult to distinguish between the VP coordination and other types of continuous clauses based only on the shallow semantic information.

3 NTT treebank

The phrase structures and supplementary information in NTT treebank (NTB) are annotated to news-wire text (Tanaka and Nagata, 2013). As supplementary annotations, the treebank contains functional tags and predicate argument structures.

Grammatical role for mandatory argument	
-SBJ	Subjective case
-OBJ	Objective case
-OB2	Indirect object case
-COORD	Coordination
-APPOS	Apposition

Table 2: Function tags in NTT treebank.

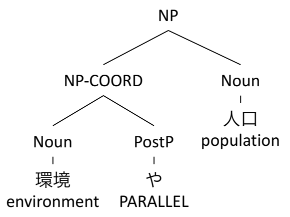


Figure 3: NP coordination in NTT treebank.

Table 2 lists some function tag examples that are annotated to the tree nodes. SBJ and OBJ represent the grammatical roles of phrases, and COORD shows the annotated node and sibling node are coordinated (Fig. 3). Predicate argument structures are presented as relations between the predicate words and their argument phrases, which is similar to the annotation style of PropBank (Palmer et al., 2005) (Fig. 5).

The treebank is created by manually correcting and updating the base annotation, which is actually the source treebank used to build the original Japanese CCGbank. It is based on the dependency between chunks or *bunsetsu* of the Kyoto corpus (Kurohashi and Nagao, 2003), but manual annotation made the treebank cleaner and richer. In addition to fixing the apparent errors such as the tokenization errors and erroneous POS tags, the manual annotation includes modifying the subtrees for specific constructions (e.g., coordinated phrases), a clause with a formal noun, and a PP with a compound postposition. Moreover, PAS annotations for specific voices, such as causatives and beneficials were added to the treebank.

Compound postposition is a type of multi-word expressions in which a combination of postpositions, verbs, and auxiliaries works as one post-

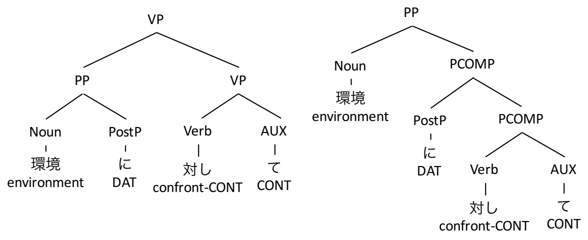


Figure 4: Subtrees with compound postposition "に対して" before (left) and after manual annotation (right).

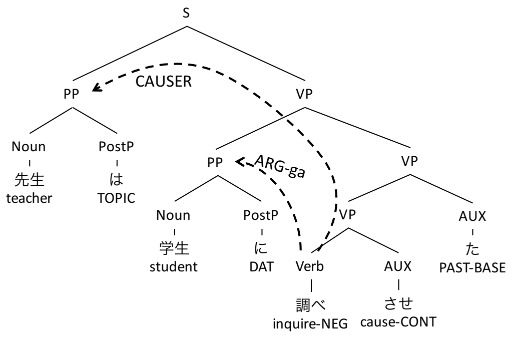


Figure 5: Causative sentence "The teacher has the student inquire" and PAS annotated for "調べ inquire". The dotted arc labeled with ARG-ga denotes that the agent of "inquire" is "the student"

position. For example, a compound postposition " に/DAT/ni 対し/confront-CONT/taishi て /aux-CONT/te" typically functions similarly to the postposition "に/DAT/ni". Fig. 4 shows the subtrees before and after the update. Since the structure on the left is the same as the structure for a continuous clause, it is difficult to distinguish compound postpositions and VPs. After the manual annotation, the compound postpositions are marked with "PCOMP" tags and have a specific structure, as shown on the right in Fig. 4

The PAS annotation on the base treebank, which is obtained by converting the word-to-word annotation of the NAIST corpus (Iida et al., 2007), was also manually corrected and populated. An important addition to the PAS is the annotation of causative and beneficial constructions. The original treebank (and the NAIST corpus) also identifies causative constructions, but there are very few annotations for the causer role, which typically occurs with the case marker "が/ga" or its topicalized form. Fig. 5 shows an example of the annotation of a causative construction with a causer role.

4 Related work

The corpus-based acquisition of wide-coverage CCG resources has been very successful for English (Hockenmaier and Steedman, 2007). Their method converts the Penn Treebank (Marcus et al., 1993) into CCGbank, which is a collection of CCG derivations, and extracts a wide-coverage lexicon from the derivations. The CCGbank is also used to train a robust CCG parser (Clark and Curran, 2007).

Complementary resources on the Penn Treebank are used to improved the derivations because the treebank does not contain some of the linguistic information necessary for a CCG derivation. Boxwell and White (2008) augmented the English CCGbank with the semantic roles in PropBank (Palmer et al., 2005). Honnibal et al. (2010) integrated several types of additional annotations such as PropBank and NP structure annotation (Vadas and Curran, 2007), to improve the CCGbank. Our work basically follows these methods, but we have to deal with noises in the treebank that are caused by the dependency-to-tree conversion errors.

Our previous work (Uematsu et al., 2015) extended the method used for the English CCGbank, and obtained wide-coverage CCG resources for Japanese. A treebank is created in this method by converting chunk-based annotation resources. The treebank is then converted into a CCGbank for Japanese, which can be used to obtain wide-coverage lexicon and parsers for Japanese CCG.

Other than the one mentioned above, there are several other studies on Japanese deep parsing. The theoretical work by Gunji (1987) describes Japanese phenomenon based HPSG. Komagata (1999) proposed a CCG-based theory and implementation, but the focus is not on processing real world texts. JACY (Siegel and Bender, 2002) is a type of hand-crafted Japanese grammar based on HPSG that can compute a detailed semantic representation. One of our future goals is to obtain CCG resources that allow for a more precise and detailed description by incorporating additional annotations into CCGbank.

5 Incorporating additional annotation into CCGbank

We describe the two steps needed to incorporate the annotations of the NTT treebank (NTB) into the Japanese CCGbank. First, we reconstruct the CCGbank according to the clean phrase structure

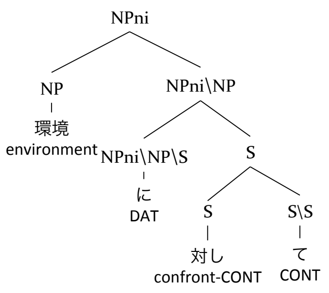


Figure 6: Substructure for argument PPs with compound postpositions.

X	CONJ	→	X_{conj}	(Coord1)
X_{conj}	X	→	X	(Coord2)

Figure 7: Special rules for coordination.

in the treebank, and then, change the substructures of the CCG derivations based on the functional tags and predicate-argument annotations of the treebank.

5.1 Reconstruction of CCGbank

As stated in Section 3, the trees in NTB are a manually corrected version of those used in (Uematsu et al., 2015). The reconstruction of the Japanese CCGbank is basically done by following the conversion rules used in (Uematsu et al., 2015). A drastic change of the conversion rules is not necessary because most of the changes in NTB are error corrections. However, we have to add a conversion rule for a compound postposition in order to handle the structure change illustrated in Sec. 3.

The treatment of compound postpositions is not explicitly described in (Bekki, 2010), so we defined two types of structures for these compounds. As with a normal postposition, a compound postposition either works as an argument marker or forms an adjunct PP. Fig. 6 shows the defined substructure for an argument PP using the compounds. The structure for case markers (left in the figure) is designed so that the node for the compound postposition (right child of the top node in the figure) is assigned the category for argument marker ($NP_{ni}\backslash NP_{nc}$ in the example).

We added a conversion rule that first detects a PP with a compound postposition by searching for a node with a PP label whose right daughter is PCOMP, and then, checks whether the node is identified as an argument by the original conversion rules, and finally assigns categories to the nodes in the PP according to the substructure shown in Fig. 6 if the node is found to be an argument.

5.2 Incorporation of the additional annotation to CCGbank

The process to incorporate complementary linguistic information into the CCGbank follows (Honnibal et al., 2010), but we have to deal with constructions that are specific to Japanese. We describe how to improve the treatment of the coordination as an example of handling functional tag annotations, and present how to identify the causer roles by processing the semantic annotations. Finally, we check the consistency of the changes.

5.2.1 Coordination

We added a new syntactic feature *conj* and two special rules that are presented in Fig. 7 to the grammar to deal with the argument coordination in derivations. The new feature indicates whether the phrase includes a conjunction. The new rules are the result of adapting the coordination rule (Φ) into binary-branching of the CCGbank. Similar rules were used to deal with the coordinations in the English CCGbank (Hockenmaier and Steedman, 2007).

We replace the analyses of the coordinations with ones incorporated with the special rules by the following process. First, a noun phrase including a coordination is detected by a subtree where an top NP node contains an NP node with a COORD tag as its left child, and the NP node with the COORD tag has a punctuation or parallel particle as its right child (the left tree in Fig. 8). The categories corresponding to the top node (a basic category NP in the example) are then checked to see if the condition for Φ is satisfied. If satisfied, the rule combining the left and right daughters is changed to Coord2. For the example in the figure, the category for "食料 / food" will be NP, and that of the left daughter will be NP_{conj} after the change. The rule to form the left daughter is also replaced by Coord1, so the category for the conjunction changes to CONJ.

5.2.2 Causer argument

The grammar for the original CCGbank can handle causer arguments as well as other types of arguments. An example analysis of a causative sentence is shown in Fig. 9. The semantic representation is omitted in this figure, but the causative verb "調べ / inquire" has a causer argument in its

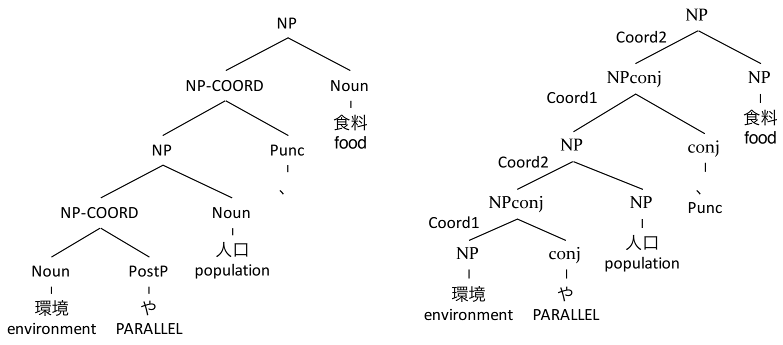


Figure 8: Subtree involving coordination (left) and new analysis for phrase (right).

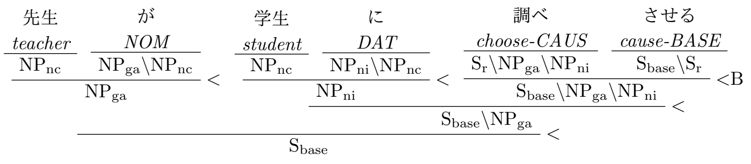


Figure 9: CCG derivation for Japanese sentence "The teacher has the student inquire."

	Train.	Devel.	Test
sentence	6,800	800	2,400
tokens	178,732	24,159	64,824

Table 3: Statistics of input linguistic resources

predicate-argument structure, and the causer argument is co-indexed with the $\mathrm{NP_{ga}}$. We reanalyzed the causative constructions in the original CCGbank based on the annotation instances of the causers added in the NTB.

First, we describe the changes to the argument phrases in the causative constructions. In Japanese, an argument to a verb is typically followed by a case marker particle ("が / ga / NOM", "に / ni / DAT", etc.) or a binding particle ("は / wa / TOPIC", "も / mo "). Phrases headed by a binding particle are used when an argument is topicalized and the case for the topicalized argument must be estimated. On the other hand, a phrase with a case marker or a binding particle can be used as a modifier to a verb. Therefore, properly distinguishing the arguments and modifiers is important for building derivations. Moreover, a causative is a construction involving case alternations (see Fig. 5), so the surface and deep cases of each argument must be decided according to the PAS annotations.

Concretely speaking, we changed the substructure for a causative in the following process. A candidate for a causer argument is detected as a phrase headed by a case marker or binding particle that is combined with a VP headed by a causative verb in the treebank. For example, the PP "先生は / teacher-TOPIC" in Fig. 5 satisfies these conditions. Next, the argument / adjunct distinction of the phrase is updated by simply checking the new PAS annotations of NTB. If the phrase is found to be a causer argument, the category for the phrase is changed to $\mathrm{NP_{ga}}$ because the surface case of the causer is always *ga*. The category for the VP is also changed to the one with the added argument. Fig. 9 shows an example of the change in causative constructions. The category for the PP "先生は / teacher-TOPIC" changes from a modifier S/S to an argument $\mathrm{NP_{ga}}$. We transfer the changes to the descendant nodes and obtain the new derivation, as shown on the right in the figure.

5.2.3 Consistency check

Finally, we check to see if the modified parts in a derivation are consistent with each other. This is done by applying the combinatory rule assigned to the each branching in the derivation in a top-down order. We discard the modifications to a derivation if they are found to be inconsistent.

6 Evaluation

We actually applied the improvement process to the Japanese CCGbank to evaluate it. The Japanese CCGbank we used was the version in December 2014. We used the preliminary version for the NTT treebank (NTB) that contained 10,000 trees with functional tags and PAS annota-

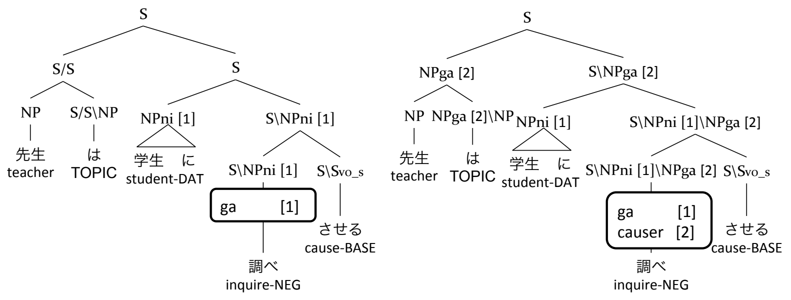


Figure 10: Causative construction with causer argument (left) and its reanalysis (right).

	Training		Development		Test	
	Annot.	Changed	Annot.	Changed	Annot.	Changed
Causative	195	–	20	–	80	–
Causer	34	22	3	3	15	10
NP Coord	2,632	2,148	323	259	826	652

Table 4: Statistics of annotated and reanalyzed instances

tions. We divided the 10,000 sentences into three sets: training, development, and test. Table 3 classifies the statistics of the sets. Since the original CCGbank consists of approximately 38,400 derivations, out experiments were performed on only 26% of the derivations.

6.1 Evaluation of the derivation changes

Table 4 lists the numbers of annotation instances and the number of the changes made in each set. We also measured the similarity between the original CCGbank and the new ones following (Honnibal et al., 2010). In other words, we used the difference in dependencies as the difference metrics, where a dependency is defined as a 4-tuple: a head of a functor, a functor category, an argument slot, and a head of an argument. Table 5 lists the percentages of the labeled and unlabeled dependencies left unchanged after the reanalysis processes. A labeled dependency is marked as unchanged if the four elements match a tuple in the original. An unlabeled dependency is correct if the heads of the functor and the argument appear together in the original.

We see from Table 5 that the most influential change was the correction of the phrase structures, which includes the changes for the compound postpositions. On the other hand, the effect of the causer arguments was fairly limited partly because there are only a small number of annotation instances for the causers.

In order to evaluate the quality of the changes, we randomly sampled fifty derivations from those

Corpus	L.Deps	U.Deps	Cat
+ Correction	81.3	86.7	85.9
+ Causer	81.0	86.6	85.7
+ NP Coord	77.9	84.1	83.2

Table 5: Rate of dependencies and categories left unchanged in development set.

Num.	Type of change
32	Change in subcategorization
19	From modifier to argument
18	Change to CONJ
11	From modifying noun NP/NP to NP
8	From S/S\NP to NP/NP\NP

Table 6: Most frequent changes in lexical categories.

that underwent the reanalysis, and manually investigated the samples. We first checked the changes in lexical categories and category dependencies, and then referred to the derivations for the cause.

6.1.1 Changes in lexical categories

The lexical categories for 135 tokens in the fifty sentences changed after the reanalysis[1]. Table 6 classifies the most frequent types of category changes.

The most and second-most frequent types are related to the adjunct versus argument decisions. Due to the corrected PASs and the additional causer annotations in the NTB, some postposition phrases previously marked as adjuncts are now arguments in the new analysis, and vice versa.

[1]We excluded tokens with different word boundaries after the reconstruction from the number.

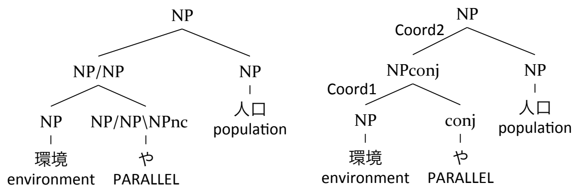


Figure 11: Subtree involving coordination (left) and new analysis for phrase (right).

In the example shown in Fig. 10, the PP "先生は / teacher-TOPIC" is changed to an argument. This has the category for "調べ / inquire-CONT" change from $S\backslash NP_{ni}$ to $S\backslash NP_{ni}\backslash NP_{ga}$, and the category for "は / TOPIC" shifted from $S/S\backslash NP_{nc}$ to $NP_{ga}\backslash NP_{nc}$ The former belongs to the most frequent type and the latter belongs to the second. These types of changes are also caused by corrections in the morphological information, e.g., POS tags. For example, a PP headed by a postposition "と" turned into an argument by fixing the erroneous POS "conjunctive particle" into the correct "case marker particle" one.

The third and fourth ones are due to the introduction of noun coordinations. For these types, the conjunctions previously treated as NP/NP\NP etc. improved to CONJ (see Fig. 11). The fourth type also resulted from corrections in the internal structures of the NPs.

We marked each of the changes in the lexical category as "good", "bad" (for deletion of obvious arguments etc.), and "cannot decide" (for cases where categories are not correct before or after the change). We found that 79% of the changes (107 categories) were judged as good.

6.1.2 Changes in dependencies

Next, we investigated the difference in dependency relations. For the fifty sampled derivations, the dependencies were extracted from the original CCGbank and our resulting CCGbank, and the two sets of dependencies were then manually compared. We focused on the dependency relations that were not shared by the two sets. In other words, we examined 348 dependency tuples unique to the original CCGbank, and 337 tuples that were only extracted from our resulting CCGbank. Tables 7 and 8 list the most frequent causes of the changes in category dependency counted in the original CCGbank and ours.

In both sets, over 90 relations only differ in the word boundaries to their counterparts. This is due to the correction of the word boundary given by the NTB, and suggests that the rate of the virtually unchanged relations is larger than those listed in Table 5.

	Original	Results
Good	207	196
Bad	34	34
Other	14	15
Total	255	245

Table 9: Judgment on unshared dependencies in original and resulting CCGbank.

CCGbank	# Cat.	Sent. cov.
Original	606	78.1
+ Correction	690	76.6
+ Causer	702	75.4
+ Coord.	693	75.2

Table 10: No. of category types and sentential coverage of lexicon extracted from each CCGbank version.

The second most frequent causes in both sets is related to the change between the adjuncts and arguments described in Sec. 6.1.1. If a PP changed from an adjunct to an argument like in the example shown in Fig. 10, the category for the postposition ("は / TOPIC" in the figure) turns into $NP_{ga}\backslash NP_{nc}$ from $S/S\backslash NP_{nc}$. This resulted in a deletion of the relations that the old category had with the left noun ("先生 / teacher" in the figure) and the main verb ("調べ / inquire "), and the addition of a relation between the new category and the left noun. The change between the adjuncts and arguments also affected the subcategorization of the predicatives, and this is counted as a change in the subcategorization (fifth in both Tables 7 and 8)

As in Sec. 6.1.1, we also marked each of the unshared dependencies as "good", "bad", or "cannot decide". We excluded the relations changed by the correction in the word boundary. Note that any dependency in the original CCGbank is marked as "good" when its deletion or change is desirable for improving the derivation. Table 9 lists the number of relations for each of the marks. More than 80% of the changes in dependency are considered to be desirable in both sets.

6.2 Evaluation of the obtained resources

6.2.1 Lexical categories

Table 10 lists the number of category types in the CCGbank and the coverage of the lexicon on the

Num.	Type of change
93	Change in word boundary
49	From verb modifier to argument
26	From modifying noun NP/NP to NP
22	Head change in argument NP
11	Change in subcategorization
11	From noun modifier to $S\backslash NP_{ga}\backslash NP$

Table 7: Most frequent types of change in category dependency (counted in original CCGbank)

Num.	Type of change
92	Change in word boundary
47	From verb modifier to argument
22	Head change in argument NP
19	Dependency for coordinated arguments
16	Change in subcategorization

Table 8: Most frequent types of change in category dependency (counted in resulting CCGbank)

	Development				Test			
	LP	LR	UP	UR	LP	LR	UP	UR
Original	84.54	81.02	90.78	87.00	85.00	81.03	91.15	86.90
+ Correction	85.05	82.24	91.33	88.32	85.24	81.44	91.99	87.89
+ Causer	84.84	82.13	91.34	88.41	84.87	81.22	91.83	87.88
+ Coord.	82.71	79.78	89.39	86.22	82.60	78.71	89.81	85.59

Table 11: Parsing accuracy

unseen text for each CCGbank version. We measured the coverage in the same way as in (Uematsu et al., 2015), that is, we obtained improved CCG derivations for the test set by applying our method to the original CCGbank, and used them as the "gold-standard". The lexical coverage was around 98.8% for all the versions. The sentential coverage indicates the number of sentences in which all the words were assigned gold-standard categories.

After applying the change to the derivations, the numbers of category types increased and the coverage dropped 2-3%. This is due to the distinctions we added to the grammar, such as the compound postpositions versus the continuous clauses.

6.2.2 Parsing accuracy

We trained a statistical parser on different versions of the augmented CCGbank, and tested on the unseen text. Table 11 itemizes the performance of the parsers trained on each version of the CCGbank. The parser we used is the same one as that used in our previous work (Uematsu et al., 2015), and no tuning was performed. The evaluation measures were the recall and precision over the category dependency. As we stated above, the size of the NTB used in this experiment was 26% of the original CCGbank, so the numbers are not directly comparable to the results using the original. Note that the table suggests how hard it is to recover the structures in each CCGbank, rather than how good the CCGbank is, because each line represents a different gold standard. A possible explanation for the slight improvement in the performance after correcting the trees is that the manual correction increased the consistency in the structures.

7 Conclusion

A method for improving the Japanese CCGbank by integrating CCG derivations and additional annotations to the source treebank was presented in this paper. For the Japanese CCGbank, the source treebank itself is a result of converting chunk-dependencies, and the treebank potentially includes more errors in the phrase structures and other types of information such as functional tags. Moreover, it essentially lacks linguistic information which is difficult to represent in the chunk-dependency, e.g., coordinated arguments.

We showed how functional and semantic annotations on the treebank can be used for improvement of the Japanese CCGbank by incorporating annotations of the NTT treebank, especially those for coordinations and causer roles. The process first reconstructs the CCGbank by using cleaner trees from the NTT treebank and the original tree-to-derivation conversion. The new derivations are then modified according to the functional tags and PAS information of the NTT treebank.

The empirical evaluation on the dependency relations shows that the improving process changed 23% of the dependencies extracted from the original CCGbank. A manual investigation of the changes suggests that approximately 80% of the changes are desirable and that the resulting derivations are more accurate in recognizing arguments and coordinations.

References

Masayuki Asahara. 2013. Comparison of syntactic dependency annotation schemata. In *Proceedings of the third Japanese Corpus Linguistics Workshop*. (In Japanese).

Daisuke Bekki. 2010. *Formal Theory of Japanese Syntax*. Kuroshio Shuppan. (In Japanese).

Stephen Boxwell and Michael White. 2008. Projecting Propbank roles onto the CCGbank. In *Proceedings of LREC 2008*.

Stephen Clark and James R. Curran. 2007. Wide-coverage efficient statistical parsing with CCG and log-linear models. *Computational Linguistics*, 33(4).

Takao Gunji. 1987. *Japanese Phrase Structure Grammar: A Unification-based Approach*. D. Reidel.

Julia Hockenmaier and Mark Steedman. 2007. CCG-bank: A corpus of CCG derivations and dependency structures extracted from the Penn Treebank. *Computational Linguistics*, 33(3):355–396.

Julia Hockenmaier. 2006. Creating a CCGbank and a wide-coverage CCG lexicon for German. In *Proceedings of COLING/ACL 2006*.

Matthew Honnibal, James R. Curran, and Johan Bos. 2010. Rebanking CCGbank for improved NP interpretation. In *Proceedings of ACL 2010*, pages 207–215.

Ryu Iida, Mamoru Komachi, Kentaro Inui, and Yuji Matsumoto. 2007. Annotating a Japanese text corpus with predicate-argument and coreference relations. In *Proceedings of Linguistic Annotation Workshop*.

Nobo Komagata. 1999. *Information Structure in Texts: A Computational Analysis of Contextual Appropriateness in English and Japanese*. Ph.D. thesis, University of Pennsylvania.

Sadao Kurohashi and Makoto Nagao. 2003. Building a Japanese parsed corpus. In *Treebanks*, volume 20 of *Text, Speech and Language Technology*, pages 249–260. Springer Netherlands.

M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Yusuke Miyao and Jun'ichi Tsujii. 2008. Feature forest models for probabilistic HPSG parsing. *Computational Linguistics*, 34(1):35–80.

Martha Palmer, Daniel Gildea, and Paul Kingsbury. 2005. The Proposition Bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31(1):71–106.

Melanie Siegel and Emily M. Bender. 2002. Efficient deep processing of Japanese. In *Proceedings of the 3rd Workshop on Asian Language Resources and International Standardization*.

Mark Steedman. 2001. *The Syntactic Process*. MIT Press.

Takaaki Tanaka and Masaaki Nagata. 2013. Constructing a practical constituent parser from a Japanese treebank with function labels. In *Proceedings of the Fourth Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 108–118.

Sumire Uematsu, Takuya Matsuzaki, Hiroki Hanaoka, Yusuke Miyao, and Hideki Mima. 2015. Integrating multiple dependency corpora for inducing wide-coverage Japanese CCG resources. *ACM Trans. Asian Low-Resour. Lang. Inf. Process.*, 14(1):1–24.

David Vadas and James Curran. 2007. Adding noun phrase structure to the Penn Treebank. In *Proceedings of ACL 2007*, pages 240–247.

David Vadas and James R. Curran. 2008. Parsing noun phrase structure with CCG. In *Proceedings of ACL-08: HLT*, pages 335–343.

Dependency Parsing with Graph Rewriting

Bruno Guillaume
Université de Lorraine, CNRS
LORIA, UMR 7503,
Vandœuvre-lès-Nancy, France
Inria, Villers-lès-Nancy, France
Bruno.Guillaume@loria.fr

Guy Perrier
Université de Lorraine, CNRS
LORIA, UMR 7503,
Vandœuvre-lès-Nancy, France
Inria, Villers-lès-Nancy, France
Guy.Perrier@loria.fr

Abstract

We propose to use Graph Rewriting for parsing syntactic dependencies. We present a system of rewriting rules dedicated to French and we evaluate it by parsing the SEQUOIA corpus.

1 Introduction

The most popular frameworks (TAG, CG, LFG, HPSG) for symbolic parsing are based on the notion of grammar. They defined a set of initial structures (often strongly linked to a lexicon) and a set of rules to express how initial structures can combine into larger ones. In this setting, parsing consists in searching for a syntactic structure predicted by the grammar for an input sentence. Among drawbacks of these methods, there is the fact that they may be inefficient when large-coverage grammar are considered (the search space grows very quickly for large sentences) and that they are not easy to use in contexts where robust parsing is needed (grammars describe set of correct sentences but do not give structures to sentences that are not completely covered by the grammar). Another problem with grammar-based parsing is that it is a difficult task to maintain the global consistency of the grammar. Moreover, development of large coverage grammar is known to be a time-consuming task.

On the other side, statistical methods build language models with learning algorithm applied to large annotated corpora. With respect to symbolic methods, it is easier to build robust parser with these methods and it is also easier to adapt a method to a new kind of corpus or to a new natural language. The main drawbacks are that good results are obtained only if large and well-annotated corpora are available. It is also difficult to improve a system: learning provides a language model which is essentially a black box which cannot be read by a human; external mechanism must be used if someone want to include linguistic knowledge in the system.

In this paper we propose a symbolic method which is defined in a Graph Rewriting (GR) framework. The output format is dependency syntax of natural language sentences. We propose to describe the parsing process as a sequence of atomic transformations starting from a list of lexical units (a tokenized sentence) to a dependency tree[1] built on the same lexical units. Each atomic transformation is described by a handcrafted rule. Then, instead of defining a grammar that describes the set of well-formed structures, we define rules which describe linguistic contexts in which a dependency relation can appear.

The rule system input is made of lemmatized and POS-tagged sentences. For the experiments in this paper, we use the SEQUOIA corpus (Candito and Seddah, 2012) (version 6.0[2]) as the gold standard. We experiment our system in two settings: on gold POS-tagged text (taken from SEQUOIA data) and on POS-tagging given by the MElt tagger (Denis and Sagot, 2012).

We use the general framework of GR where each transformation is given by two parts: first, the conditions that control when the transformation may apply (the *pattern*) and second, a description of the way the structure should be modified.

In the system we proposed, the input format is a tokenized sentence where each lexical unit is given a lemma and a POS-tag; the output format is a dependency structure; hence, both input and output structure can be represented as trees. Nevertheless, we use GR to describe our rules. At a first sight, it may be surprising to use such a formalism to manipulate only trees. But, the first thing to notice is that matching algorithms used in GR are direct generalization of matching algorithm that can

[1]The output may a partial dependency trees.

[2]https://gforge.inria.fr/projects/sequoiabank/

be used in tree transformation: it means that, if all structures and patterns happen to be trees, the pattern matching in the GR setting will be as efficient as the pattern matching in the tree rewriting setting. A second benefit of GR is that it becomes possible to express more information in the intermediate structures. In the rule system, we use two kinds of relation to express linear order between lexical entities: the relation `SUC` links the heads of two partial dependency structures; the relation `INIT_SUC` links two successive lexical unit of the sentence, even if they have also been integrated in partial dependency structures. Structures with these two kinds of relations are graphs and cannot be represented as trees.

In a comparison with other works from the literature, we left out data-driven approaches which are far from our proposal. In (Foth et al., 2000), (Debusmann et al., 2004), weighted rules are used to described valid dependency structures and the parsing is expressed as a constraints resolution problem. (Covington, 2000) and (Nivre, 2003) propose rule-based processes to produce dependency structures but they are presented as kind of shift-reduce algorithm where word are treated one by one following the reading order, rules describing how each word can be link to the current state. Each rule only tells that a dependency from a word to another word is acceptable.

More close to our work is the proposal of (Oflazer, 2003) which defines a set of rules that are used iteratively until a fixpoint is reached and the rules application do not necessarily follow the reading order of the sentence. However, in (Oflazer, 2003) rules are encoded as regular expressions and are less flexible than a GR rule can be. To our knowledge, our proposal is the first use of the Graph Rewriting framework for symbolic dependency parsing.

In Section 2, we describe more precisely the GR framework used in the paper. In Section 3, the GR system considered is detailed. We finally give experimental results in Section 4.

2 Graph Rewriting

Unfortunately, there is no canonical formal GR definition. In this experiment, we use the GR definition which is implemented in the GREW software[3]. Rules are defined by two parts: a pattern and a set of commands. GREW was used for in-

[3]`http://grew.loria.fr`

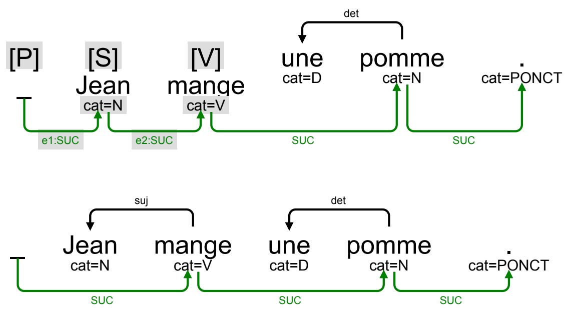


Figure 1: An example of application for the rule `subject_noun`

stance in (Bonfante et al., 2011) to build a semantic annotation of a French Treebank.

The reader can refer to the GREW documentation for a complete description of the GR framework and of the syntax of the rules with GREW. We give here a simple example of rule and of its application; more elaborated rules are shown in the next section. The code below is a simplified version of a rule for the subject relation.

```
 1 rule subject_noun {
 2   match {
 3     S [cat=N|PRO];
 4     V [cat=V, m=ind|subj];
 5     e1:S -[SUC]-> V;
 6     P []; e2:P -[SUC]-> S;
 7   }
 8   without { V -[suj]-> * }
 9   without { S [lemma="que"|"dont"] }
10   commands {
11     del_edge e1;
12     del_edge e2;
13     add_edge V -[suj]-> S;
14     add_edge P -[SUC]-> V;
15   }
16 }
```

Different parts of rule are as follows. One **`match`** part (lines 2 to 7) describes the subgraph that must be found: here, the subgraph contains 3 nodes `S`, `V` and `P` linked by two relations `SUC` (lines 5 and 6). Any number of **`without`** parts describe negative application patterns; if any of these negative patterns is found, the rule application is blocked. For instance (line 8), if there is already a `suj` dependency starting from `V`, the rule does not apply (it prevents from putting two subjects for the same verb). Finally, a **`commands`** part describes how the graph is transformed by the rule application, in the example: add a `suj` relation from the `V` node to the `S` node and update the `SUC` relations (the verb is now the successor of the `P` node). On a very simple sentence *Jean mange une pomme. (John eats an apple.)*, the two dependency structures of Figure 1 show respectively the structure before (with identifiaction of matched nodes `S`, `V` and `P`) and after the rule application. With GREW, it is also possible to modify feature structures in the **`commands`** part: add a new feature, modify or remove an existing feature. The rule above uses some lexical information in the second **`without`** part. As some rules make a strong usage of lexical information, it is possible to parametrize rules by external lexicons with the **`lex_rule`** keyword. For instance, in the rule below, a relation `obj` in added (line 18) only if the verb lemma (line 7) is one on the lemmas given in the lexical file `verb_with_obj_noun.lp` (line 3). This file contains a list of more than 3,000 French transitive verbs.

```
 1  lex_rule verb_object_noun
 2    ( feature $lemma;
 3      file "verb_with_obj_noun.lp"
 4    ) {
 5    match {
 6      OBJ [cat=N|PRO];
 7      V [cat=V, lemma=$lemma, m=ind|subj];
 8      POST [];
 9      e1: V -[SUC]-> OBJ;
10      e2: OBJ -[SUC]-> POST;
11      V -[suj]-> *
12    }
13    without ...
14    without ...
15    commands {
16      del_edge e1;
17      del_edge e2;
18      add_edge V -[obj]-> OBJ;
19      add_edge V -[SUC]-> POST;
20    }
21  }
```

When the number of rules increases, some constraint should be put on the order in which the rules can be used. In this respect, a last feature, which is essential in the GREW usage, is the organization of rules into subsets called *modules*. The whole rewriting process is controlled by a total order on modules that are applied one after another; inside a module, no ordering is given and any rule may be apply anywhere in the graph.

Moreover in a general GR setting, the process may be non-confluent. Even if a total ordering is provided between modules, an arbitrary number of structures can be produced inside a module. In this work, we restrict ourself to deterministic GR, this means that we focus on the set of dependency relation that can be produced in a deterministic way. We can imagine many cases where non-confluent rewriting system can be used: for instance, PP-attachment is known to be a task that cannot be decided only with syntactic information. We leave as future work the development of non-confluent system and the problem of ranking in case of non-deterministic process.

3 FRDEP-PARSE: a System of Graph Rewriting Rules for Dependency Parsing

3.1 Input and Output Formats

FRDEP-PARSE takes a French sentence annotated with POS-tags and lemmas as input and returns a dependency syntax annotation for the same sentence as output. Let us describe the input and the output more precisely.

The input sentence is a sequence of tokens. Each token is equipped with a set of features:

- `phon`: the phonological form of the token;
- `lemma`: the lemma of the token;
- `pos`: the value of which is one of the 28 tags defined in the annotation guide of the French TreeBank[4];
- `position`: an integer indicating the position of the token in the sequence (this feature is used to express linear order between tokens).

At the beginning of the process, the 28 POS-tags are interpreted in terms of 13 grammatical categories (feature `cat`) and some other features. For instance, the 6 POS-tags `V`, `VINF`, `VIMP`, `VS`, `VPP`, `VPR` are all interpreted by feature `cat=V` and a `m` feature recording the mood (respectively indicative, infinitive, imperative, subjunctive, past participle and present participle).

Initially, the only relations between tokens are `SUC` relations of immediate succession between adjacent tokens.

The parsing output is the sentence annotated with syntactic dependencies according to the tagset used in SEQUOIA. The annotation may be partial. Figure 2 shows an example of syntactic annotation obtained with FRDEP-PARSE.

3.2 A CKY Basic Architecture

The basic form for the rewriting rules of FRDEP-PARSE aims at the implementation of a CKY-like algorithm in the dependency syntax framework. The dependency tree of a sentence is built bottom-up step by step. When two partial trees T_1 and T_2 have their terminal yields which are adjacent, a dependency may be added between the roots w_1 and w_2 of the two trees.

[4] `http://alpage.inria.fr/statgram/frdep/Publications/FTB-GuideDepSurface.pdf`

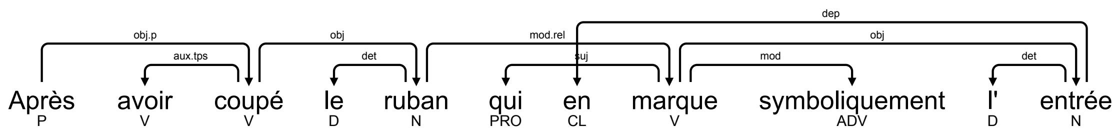


Figure 2: A non-projective result of syntactic annotation with FRDEP-PARSE.

To express adjacency between two yields, `SUC` is used with a larger meaning: if there is a `SUC` relation from a token w_1 to a token w_2, it means that w_1 and w_2 are roots of intermediate dependency trees, the yield of the first tree immediately preceding the yield of the second tree.

(Eisner, 1996) already proposed a CKY-like algorithm for parsing with dependencies, but he differs from our proposal on two points: he uses a statistical approach and to link two dependency trees, he takes only their roots into account and ignores information coming from deeper nodes.

The use of GREW for implementing the CKY algorithm in a strict manner has no point, but the Graph Rewriting approach allows to enrich the algorithm in various directions. One of them is to add internal and external constraints on the T_1 and T_2 trees.

Here is an example of constraints introduced by a rule of FRDEP-PARSE.

```
lex_rule verb_right-modif_adv
 (feature $lem; file "quant_adv.lp") {
  match {
    %positive constraint on T1
    V [cat=V];

    %positive constraints on T2
    ADV[cat=ADV];
    POST [ ];
    e1: ADV -[SUC]-> POST

    % adjacency between T1 and T2
    e2: V -[SUC]-> ADV
  }

  % negative constraints on T1
  without { ADV[pos=ADVWH] }

  %negative constraints on T2
  without { POST[cat=V,m=pastp] }
  without {
    ADV[lemma=$lem];
    POST[cat=P,lemma="de"]
  }

  commands{
    del_edge e1;
    del_edge e2;
    add_edge V -[mod]-> ADV;
    add_edge V -[SUC]-> POST
  }
}
```

The rule above is a lexical rule in which the parameter `$lem` represents the lemma of an adverb coming from the `quant_adv.lp` file gathering adverbs of quantity. It says that any adverb `ADV`, the yield of which immediately follows the yield of a verb `V`, is a modifier of the verb `V`. It includes three negative constraints:

- An internal constraint: `ADV` must be different from an interrogative adverb. For instance, in the sentence *Pierre demande combien ça coûte (Pierre asks how much it is)*, *combien* is not a modifier of *demande* but a complement of *coûte*.
- An external constraint: `POST` that immediately follows `ADV` is not a past participle because in this case, `ADV` depends on the past participle `POST`. For instance, in the sentence *Pierre a beaucoup travaillé (Pierre has worked a lot)*, *beaucoup* is not a modifier of the auxiliary *a* but of the past participle *travaillé*.
- A mixed constraint: `ADV` is not an adverb of quantity followed with the preposition *de*. For instance, in the sentence *Pierre connaît beaucoup de personnes (Pierre knows a lot of persons)*, *beaucoup* is not a modifier of *connaît* but of *personnes* at the opposite to the sentence *Pierre travaille beaucoup la nuit (Pierre works a lot in the night)*.

The following example highlights the expressivity of the graph rewriting approach, which allows the representation of constraints on deep levels in the T_1 and T_2 trees .

```
lex_rule impers_verb_obj_de_inf
 (feature $lem;
  file "il_verb_with_obj_de_inf.lp"){
  match {
    %positive constraints on T1
    V[cat=V,lemma=$lem];
    IL[phon="il"|"Il"];
    V -[suj]-> IL;

    %positive constraints on T2
    PREP[cat=P,lemma="de"];
    OBJP[cat=V,m=inf];
    PREP -[obj.p]-> OBJP;
    POST[];
    e1: PREP -[SUC]-> POST;

    % adjacency between T1 and T2
    e2: V -[SUC]-> PREP;
  }

  % negative constraints on T1
  without{SE[pos=CLR]; V -> SE}

  commands{
    del_edge e1;
    del_edge e2;
    add_edge V -[obj]-> PREP;
    add_edge V -[SUC]-> POST
  }
}
```

The rule above realizes the direct object of an impersonal verb, when this object is an infinitive introduced with the preposition *de*. The rule is lexical because it verifies that the verb is able to enter such a construction. The parameter of the rule is the lemma `$lem` of the verb.

A positive constraint on T_1 says that the verb `V` must have a subject *il* and a positive constraint on T_2 says that the tree is made of `PREP` *de*, a preposition introducing an infinitive `OBJP`.

Such constraints cannot be expressed with the classical CKY algorithm in a constituency approach, where constraints are limited to the external constituents of partial syntactic trees.

3.3 Non Projective and Disambiguation Rules

If all rules of FRDEP-PARSE had the form described above, the dependency structure resulting from their application would be projective. Therefore, we should be not able to represent some phenomena from the French grammar, which are essentially non-projective, and are present in the SEQUOIA corpus.

In FRDEP-PARSE, non-projectivity is introduced in two ways:

- in iterative modules, some rules link partial dependency trees with non-projective dependencies;
- specific rules are added in final modules with the aim of moving dependencies from provisional positions with provisional labels in order to obtain non-projective dependencies with definitive labels.

There are other final modules used for disambiguation. Indeed, at some intermediate steps of the parsing process, there is no sufficient information for deciding between several dependency labels. We could divide the search path into several paths. The repetition of such choice points would entail the time explosion of the parsing process. In this situation, we keep a unique search path by labelling the concerned dependencies with disjunction of elementary functions. At the end of parsing, the ambiguity is solved with specific rules using the information accumulated during the parsing process.

The following rule illustrates both functions: label disambiguation and expression of non-projectivity. It applies to Sentence *[annodis.er_00026]* from the SEQUOIA corpus, more specifically to the part *le ruban qui en marque symboliquement l'entrée (the ribbon, which symbolically marks the entry of it).* Figure 1 shows the syntactic annotation produced by FRDEP-PARSE. The clitic pronoun *en* represents a complement of *entrée*. It is modelled with a dependency `dep` crossing the object dependency `mod.rel` from the noun *ruban* to the verb *marque*. The rule producing non-projectivity is the following:

```
lex_rule obj_dep_en
 (feature $lem ;
   file "verb_with_obj_deobj.lp") {
  match{
    CLIT[pos=CLO, phon="en"];
    V[cat=V];
    iobj_rel: V -[DE_OBJ-OBJ]-> CLIT
    OBJ[cat=N];
    V -[obj]-> OBJ
  }
  without { V [lemma=$lem] }
  commands {
    del_edge iobj_rel;
    add_edge OBJ -[dep]-> CLIT
  }
}
```

Initially, `CLIT` *en* depends on the verb `V`, which it cliticizes in an ambiguous relation `de_obj` (indirect object introduced with the preposition *de*) or `obj` (direct object). When the verb `V` has found a direct object `OBJ`, the source of the dependency for `CLIT` *en* is transferred from the verb to the object and its label becomes `dep`.

The lexicalization of the rule concerns the negative constraint. It aims at verifying that `V` is not a verb simultaneously taking a direct object and an indirect object introduced with *de*. In this case, *en* would be the `de_obj` complement of the verb `V`.

3.4 Modules for Controlling the Parsing Process

If we put all rewriting rules of FRDEP-PARSE in a unique bag with the same priority, the system is untractable because of the ambiguity, which will entail an time explosion of the parsing process. But GREW offers the possibility of grouping rules by modules and ordering the modules.

We use this possibility for controlling the pars-

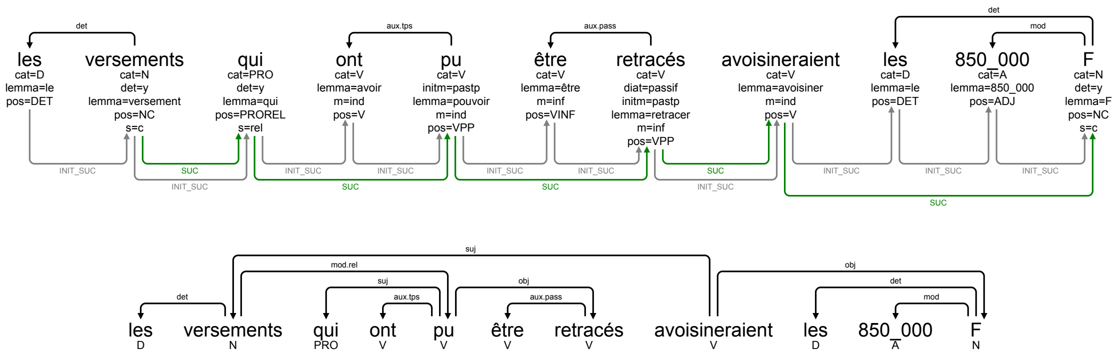

det
aux.tps
aux.pass
det
mod
les cat=D lemma=le pos=DET
versements cat=N det=y lemma=versement pos=NC s=c
qui cat=PRO det=y lemma=qui pos=PROREL s=rel
ont cat=V lemma=avoir m=ind pos=V
pu cat=V initm=pastp lemma=pouvoir m=ind pos=VPP
être cat=V lemma=être m=inf pos=VINF
retracés cat=V diat=passif initm=pastp lemma=retracer m=inf pos=VPP
avoisineraient cat=V lemma=avoisiner m=ind pos=V
les cat=D lemma=le pos=DET
850_000 cat=A lemma=850_000 pos=ADJ
F cat=N det=y lemma=F pos=NC s=c
INIT_SUC
SUC
INIT_SUC
INIT_SUC
INIT_SUC
SUC
INIT_SUC
INIT_SUC
SUC
SUC
INIT_SUC
INIT_SUC
INIT_SUC
INIT_SUC
SUC
suj
mod.rel
suj
aux.tps
obj
aux.pass
obj
det
mod
det
les D
versements N
qui PRO
ont V
pu V
être V
retracés V
avoisineraient V
les D
850_000 A
F N

Figure 3: The annotation of a sentence after running the initial modules and at the end of parsing.

ing process. The rewriting rules model particular grammatical rules of the language (French in our case) and they are grouped by modules according to their linguistic proximity. We distinguish three classes of modules with respect to their position in the parsing process: initial modules, iterative modules and final modules.

3.4.1 Initial modules

Initial modules are carried out at the beginning of parsing. They have two different functions. Some modules, which are related to the specific annotation format, are used to prepare the actual syntactic annotation.

Other modules have a linguistic function: they realize close dependencies to verbs, nouns, adjectives and adverbs. They realize them with a great determinism, and thus they can be very well foreseen. They link verbs with their auxiliaries and clitics, nouns with their determiner and left adjectives. Finally, they make specific adverbs modifiers of verbs, adjectives or other adverbs.

Figure 3 shows an example of annotation step produced with FRDEP-PARSE on a part of Sentence *[frwiki_50.1000_00854]* from the SEQUOIA corpus. The first annotation is produced by the application of the initial modules. The module verb_aux realizes the dependencies `aux.tps` and `aux.pass` of the respective verbs *pu* and *retracés*. The module noun_dep realizes the dependencies `det` and `mod` for the noun *F* but also the dependency `det` for the noun *versements*.

3.4.2 Iterative modules

Iterative modules realize arguments of verbs, adjectives, nouns and adverbs, as well as their modifiers that can be put more or less far from them. They are called iterative because they can be repeated several times, which is made necessary by the CKY form of the syntactic composition induced by the form of rules and the recursivity of the syntactic composition for natural languages.

In the bottom part of Figure 3, the `suj`, `obj` and `mod.rel` dependencies of the second annotation are realized by iterative modules. In particular, the subject module is used twice: first, it realizes the subject *qui* of *pu*; second, it realizes the subject *versements* of *avoisineraient*. With a CKY strategy, both dependencies cannot be realized in the same application of the module subject because *versements* must be composed with the relative clause that modifies it, before being composed with the verb *avoisineraient*.

There is a specific coord module, which is dedicated to coordination. It is also iterative because coordination may be performed at more or less deep levels of syntax: word, noun phrase, clause...

3.4.3 Final modules

Final modules are carried out at the end of parsing. There are three classes of final modules.

In the order of their execution, the first class includes modules that realize disambiguation and transformation of projective trees into non-projective trees (see Subsection 3.3).

Then, a second class includes modules that aim at closing the dependency structure of the sentence in two ways: adding default dependencies between non-connected partial trees and removing relations that are not syntactic dependencies but that were used by FRDEP-PARSE in intermediate steps.

Finally, a last class of modules transforms the annotation in a format conform to the annotation scheme of SEQUOIA. The reason is that our linguistic choices of annotation differ from those made for annotating SEQUOIA on some points.

First, we aim at simplifying the system of rewriting rules with a more uniform representation of dependencies. For instance, all contractions between a preposition and a determiner (for instance *à le* is contracted in *au*) are decomposed, so that rules related to determiners or prepositions can apply to these cases in the same way as in the standard cases.

Second, for some phenomena, there are good arguments for two different interpretations and we have made another choice than SEQUOIA. For instance, we consider that the head of a coordination is the conjunction of coordination. The main argument is that it allows the modifiers or arguments of a coordination to be distinguished from those of the first conjunct, which is not possible if we choose the head of the first conjunct as the head of the coordination.

3.5 Relaxing the CKY strategy

The goal of grouping rewriting rules by modules and ordering these ones, is the efficiency of parsing. The challenge is to keep accuracy at the same time. For this, we can play with two factors:

- the delimitation of modules (between two extremes, all rules in one module and one separate module for each rule),
- the order between modules, taking into account that iterative modules can be repeated as many times as needed,

If we constrain all iterative rules to respect the form described in Subsection 3.2, which induces a CKY strategy of parsing, we cannot obtain some needed dependency structures: for most iterative modules, the constraints imposed by the French grammar are contradictory.

Let us take again Example 1 to illustrate this contradiction. The first dependency `obj` from *coupé* to *ruban* must be realized after the dependency `mod.rel`, which entails the order head_verb_obj / head_noun_modrel between the corresponding modules. At the opposite, the second dependency `obj` from *marque* to *entrée* must be realized before the dependency `mod.rel`, which entails the order head_noun_modrel / head_verb_obj .

The contradiction cannot be solved by iteration of the module head_verb_obj because both dependencies will be realized at the first pass through the module.

We choose to relax the CKY strategy, which allows to link two partial dependency trees only by their roots. From empirical considerations, we propose new rules in the following form: if the yield of a partial dependency tree T_1 immediately precedes the yield of another partial dependency tree T_2, the rule tries to realize a dependency between the rightmost token wr_1 of the T_1 yield with the root of T_2.

This new kind of rules aims at implementing a strategy of parsing that gives priority to closest dependencies, contrary to the rules implementing a CKY-like strategy aiming at linking heads of dependency trees. We call the first rules *close linking rules* and the second ones *head linking rules*. The corresponding modules are respectively called *close linking modules* and *head linking modules*.

As for the head linking rules, the adjacency between T_1 and T_2 in a close linking rule is expressed with a `SUC` relation between their roots r_1 and r_2. The difference is that an `INIT_SUC` relation expresses the immediate succession between wr_1 and the left border wl_2 of the T_2 yield. The token wl_2 is linked to r_2 by an explicit chain of dependencies, which may reduce to the empty chain. The rule introduces a dependency from wr_1 to r_2.

Let us illustrate this new kind of rules with Example 1. Using only head linking rules, we obtain the annotation shown on Figure 4. We fail to link the word *ruban* with the head *marque* of the relative clause because the concerned module head_noun_modrel, the function of which is to realize `mod.rel` dependencies, comes after the module head_verb_obj realizing the dependency `obj` from *coupé* to *ruban*.

To solve the problem, we introduce the following close linking rule:

```
rule close-noun_modrel_verb {
  match {
    % positive constraints on T1
    PRE [];
    N [cat=N|PRO];

    % positive constraints on T2
    PROREL [pos=PROREL];
    V [cat=V, m=ind|subj];
    POST [];
    V -> PROREL;
    e2: V -[SUC]-> POST;

    %adjacency between T1 and T2
    e1: PRE -[SUC]-> V;

    % N rightmost token in T1 yield
    N -[INIT_SUC]-> PROREL
  }
```

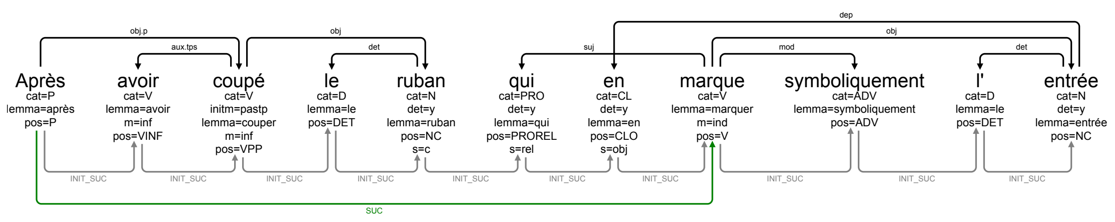


Figure 4: The annotation of the phrase from Figure 1 with the CKY strategy

```

  % PROREL leftmost token in T2 yield
  without {
    DEP []; V -> DEP;
    DEP.position < PROREL.position
  }
  without { V -> N}

  commands {
    del_edge e1;
    del_edge e2;
    add_edge N -[mod.rel]-> V;
    add_edge PRE -[SUC]-> POST;
  }
}
```

The rule links a noun N with the head verb V of a relative clause in a mod.rel dependency, expressing the modification of the noun by the relative clause.

The token PRE is a node of T_1, without any specific features. N is also a node of T_1 but the relationship between PRE and T_1 is not specified. Some constraints will be added further to make PRE the root of T_1 and N the rightmost token of its yield. The tree T_2 is made of the verb V governing a relative pronoun PROREL.

Any rule from an initial or iterative module preserves the following property for the SUC relation: the two tokens linked by the relation are roots and they have adjacent yields. This entails that PRE and V are roots of respectively T_1 and T_2 and their yields are adjacent.

As the initial and iterative modules preserve projectivity, the negative constraints express that PROREL is the leftmost token wl_2 of the T_2 yield.

Since the relation INIT_SUC expresses the adjacency of two tokens, therefore PROREL is the immediate successor of N. As PROREL is the leftmost token of the T_2 yield and the T_1 yield immediately precedes the T_2 yield, it entails that the rightmost token wr_1 of the T_1 yield is N.

The rewriting commands link the head V of the relative clause with its antecedent N in a mod.rel relation and they update the SUC relations.

As the experimental results shows it in the next section, both head linking and close linking strategies capture most configurations of syntactic dependencies but they fail to parse such phrases as the following one extracted from *[frwiki_50.1000_00464]*: *des listings de comptes de la chambre de compensation luxembourgeoise*. If we ignore agreement constraints, the head linking strategy leads to binding of the adjective *luxembourgeoise* with the head *listings* of the dependency tree of *des listings de comptes de la chambre de compensation*. With the close linking strategy, binding is realized with the closest leaf *compensation*. Both solutions are wrong because *luxembourgeoise* is a modifier of *chambre*. We need a more flexible strategy to cover all different cases of linking between two partial dependency trees.

4 Experimental results

The gold standard data used for evaluation is SEQUOIA version 6.0, which contains 3,099 French sentences taken from various sources (newspaper, medical texts, Europarl and Wikipedia). The full corpus was divided in two homogeneous subcorpora (DEV-SEQUOIA and TEST-SEQUOIA) of the same size. We used the DEV-SEQUOIA corpus to develop and to improve the rule system; the final evaluation reported below being done on the other part TEST-SEQUOIA.

The input of our rule-based system FRDEP-PARSE are sentences which are tokenized, lemmatized and tagged with the refined system of 28 pos labels defined in (Candito et al., 2011). In order to evaluate the FRDEP-PARSE rule system alone, we have made a first experiment (called GOLD-POS) where the input data are taken from the gold corpus: we consider the tokenization, lemmatization and enriched POS of SEQUOIA version 6.0 as input. The second experiment tries to evaluate our proposal in a more realistic setting where no gold tagging is available. In this second case, we first use a French tagger to build the input of out system from the raw test. Our tests are based on MElt (Denis and Sagot, 2012), so we call this second experiment MELT-POS.

In SEQUOIA, there is no clear rules in the annotation guide which explain how punctuation sign should be linked to the rest of the sentence; hence, the punctuation is not annotated in a consistent way. Here, we report scores on the Corpus TEST-SEQUOIA without taking into account the relation punctuation; nevertheless, comma used as a coordination in enumeration are annotated with relation `coord` and `dep.coord` and so there are included in the evaluation. In this context, the LAS (Labelled Attachment Score) corresponds to what is usually called the recall (*i.e.* the proportion of relations in the reference corpus which are correctly predicted by the system). The objective of FRDEP-PARSE is not to build complete dependency parse and when it is not sensible to compute a relation with a deterministic rewrite rule, only partial dependency structures are returned and some lexical units are left unattached. This explain why the precision (*i.e.* the proportion of relations predicted by the system which are correct) is much higher than the recall.

	LAS(recall)	prec.	F-measure
GOLD	80.61%	89.21%	84.69%
MELT	76.04%	85.96%	80.69%

Figure 5: Experimental results

The TEST-SEQUOIA corpus contains 1,550 sentences and 33,662 lexical units. This corpus is parsed in 51.9 seconds (with a 2.4GHz Intel Core i7) and 32,035 relations are produced; this corresponds to a mean of 617 relations produced per second.

We provide below some comparison with other dependency parsers evaluated on French.

The talismane parser (Urieli, 2013) is a statistical parsed where some rules based on linguistic knowledge can be used to guide the parser. For the same reason we gave above about punctuation, Talismane is evaluated on the set of dependencies different from the punctuation. On a corpus similar to the one we used, the LAS (labelled attachment score) ranges from 86% to 88%.

In (Villemonte De La Clergerie, 2014), some experiments on the SEQUOIA corpus are also reported. Again, results are given without taking into account the punctuation. Different combination of parsers are tested on different sub-corpora; the LAS scores obtained range from 83.53% to 88.94%.

The scores we obtain with FRDEP-PARSE are significantly lower than the two other systems but we still consider them as very encouraging. Indeed, the FRDEP-PARSE system was written in a few weeks and is the first attempt to write of a set of graph rewriting rules for dependency parsing. Moreover, we make two very strong restrictions on our system: we consider only deterministic application of rules and we do not use any statistical information. Relaxing these restrictions is far from being an easy task. If we consider non-deterministic uses of Graph Rewriting in the general setting, we will necessarily have to deal with exponential number of solutions. It will be needed to use statistical information to guide the graph rewriting process and to avoid exponential explosion. This challenge is part of the future work to do in this area.

5 Conclusion

In this paper, we have presented a first attempt to build a dependency parser in the framework of Graph Rewriting. Despite severe restrictions on the rule system, we obtain a LAS-score of 80% if the POS-tagging is taken from the Gold standard corpus and 76% if we use MElt as the POS-tagger. These results are lower than state-of-the-art approaches based on combination of statistical and symbolic dependency parsing but we think that there are many possible ways of improvement from this first attempt; for instance, using non-deterministic graph rewriting rules together with a selection system based on statistical information taken from a parsed corpus is one of the future plan for improving our system.

References

Guillaume Bonfante, Bruno Guillaume, Mathieu Morey, and Guy Perrier. 2011. Modular Graph Rewriting to Compute Semantics. In Johan Bos and Stephen Pulman, editors, *9th International Conference on Computational Semantics - IWCS 2011*, pages 65–74, Oxford, United Kingdom, January.

Marie Candito and Djamé Seddah. 2012. Le corpus Sequoia : annotation syntaxique et exploitation pour l'adaptation d'analyseur par pont lexical. In *Proc. of TALN*, Grenoble, France.

Marie Candito, Benoît Crabbé, and Mathieu Falco, 2011. *Dépendances syntaxiques de surface pour le français (version 1.2).*

Michael A. Covington. 2000. A fundamental algorithm for dependency parsing. In *In Proceedings of the 39th Annual ACM Southeast Conference*, pages 95–102.

Ralph Debusmann, Denys Duchier, and Geert-Jan Kruijff. 2004. Extensible dependency grammar: A new methodology. In *Recent Advances in Dependency Grammars*, August.

Pascal Denis and Benoît Sagot. 2012. Coupling an annotated corpus and a lexicon for state-of-the-art POS tagging. *Language Resources and Evaluation*, 46(4):721–736.

Jason M. Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th Conference on Computational Linguistics - Volume 1*, COLING '96, pages 340–345. Association for Computational Linguistics.

Kilian Foth, Ingo Schröder, and Wolfgang Menzel. 2000. A transformation-based parsing technique with anytime properties. In *in proceedings of the 4th International Workshop on Parsing Technologies*, pages 89–100.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT*, pages 149–160.

Kemal Oflazer. 2003. Dependency parsing with an extended finite-state approach. *Computational Linguistics*, 29:515–544.

Assaf Urieli. 2013. *Robust French syntax analysis: reconciling statistical methods and linguistic knowledge in the Talismane toolkit*. Ph.D. thesis, Université de Toulouse II le Mirail.

Éric Villemonte De La Clergerie. 2014. Jouer avec des analyseurs syntaxiques. In *TALN 2014*, Marseilles, France, July. ATALA.

Semantic Parsing for Textual Entailment

Elisabeth Lien
Department of Informatics
University of Oslo, Norway
elien@ifi.uio.no

Milen Kouylekov
Department of Informatics
University of Oslo, Norway
milen@ifi.uio.no

Abstract

In this paper we gauge the utility of general-purpose, open-domain semantic parsing for textual entailment recognition by combining graph-structured meaning representations with semantic technologies and formal reasoning tools. Our approach achieves high precision, and in two case studies we show that when reasoning over n-best analyses from the parser the performance of our system reaches state-of-the-art for rule-based textual entailment systems.

1 Background and Motivation

There is a growing interest in recent years in general-purpose semantic parsing into graph-based meaning representations, which provide greater expressive power than tree-based structures. Recent efforts in this spirit include, for example, Abstract Meaning Representation (Banarescu et al., 2013), and Semantic Dependency Parsing (SDP) (Oepen et al., 2014; Oepen et al., 2015). Simultaneously, in the Semantic Web community, a range of generic semantic technologies for storing and processing graph-structured data has been made available, but these have not been much used for natural language processing tasks. We propose a flexible, generic framework for precision-oriented Textual Entailment (TE) recognition that combines semantic parsing, graph-based representations of sentence meaning, and semantic technologies.

During the decade since the TE task was defined, (logical) inference-based approaches have made some important contributions to the field. Systems such as Bos and Markert (2006) and Tatu and Moldovan (2006) employ automated proof search over logical representations of the input sentences. Other systems, such as Bar-Haim et al. (2007), apply transformational rules to linguistic representations of the sentence pairs, and determine entailment through graph subsumption. Because inference-based systems are vulnerable to incomplete knowledge in the rule set and errors in the mapping from natural language sentences to logical forms or linguistics representations, and because the definition of the TE task encourages a more relaxed, non-logical notion of entailment, the majority of TE systems have used more robust approaches, however. Our work supports a notion of logical inference for TE by reasoning with formal rules over graph-structured meaning representations, while achieving results that are comparable with robust approaches.

We use a freely available, grammar-driven semantic parser and a well-defined reduction of underspecified logical-form meaning representations into variable-free semantic graphs called Elementary Dependency Structures (EDS) (Oepen and Lønning, 2006). We capitalize on a pre-existing storage and search infrastructure for EDSs using generic semantic technologies. For entailment classification, we create inference rules that enrich the EDS graphs, apply the rules with a generic reasoner, and use graph alignment as a decision tool.

To test our generic setup, we perform two case studies where we replicate well-performing TE systems, one from the Parser Evaluation using Textual Entailments (PETE) task (Yuret et al., 2010), and one from SemEval 2014 Task 1 (Marelli et al., 2014). The best published results for the PETE task, Lien (2014), were obtained through heuristic rules that align meaning representations based on structural similarity. Lien and Kouylekov (2014) extend the same basic approach for SemEval 2014 by including lexical relations and negation handling. We recast the handwritten heuristic rules from these systems as formal Semantic Web Rule Language (SWRL) rules, and run them with a generic reasoning tool over EDS meaning representations. The PETE contribution of Lien (2014) experimented with using n-best analyses from the parser to boost TE recall, and we can easily include n-best reasoning in our setup.

In Sections 2 and 3, we outline our approach and describe the semantic parsing setup and semantic technologies we employ. Sections 4 and 5 detail our replication of the two TE shared tasks. Finally, in Section 6, we sum up our effort and point to directions for future work.

2 General-purpose Semantic Parsing

General-purpose, open-domain semantic parsing systems that output logical-form meaning representations are freely available today, but have not yet been widely used in TE systems. For our replication of the PETE and SemEval tasks, we use the English Resource Grammar (ERG) (Flickinger, 2000), a broad-coverage HPSG-based parser. The ERG has been continuously developed since around 1993, and today will typically allow parsing of 90-95% of the sentences in naturally occuring running texts of various domains and genres at average parse times of a couple of seconds per sentence. The ERG includes a Maximum Entropy parse ranking model that is trained on some 50,000 mixed-domain sentences; the parser applies exact inference, i.e., constructs a complete parse forest and facilitates extraction of n-best lists of analyses in globally optimal rank order. In our experiments, we use the ERG in its 1212 release version, together with its standard PET parser (Callmeier, 2002), and off-the-shelf models and settings. The ERG outputs underspecified meaning representations in the Minimal Recursion Semantics (MRS) framework (Copestake et al., 2005). The MRS logical-form meaning representations can be converted to EDSs, which are variable-free semantic dependency graphs. Kouylekov and Oepen (2014) recently showed that the Resource Description Framework (RDF) is suitable for representing various types of semantic graphs, and demonstrated how to embed EDS meaning representations in RDF. We opt for EDS over MRS because its variable-free form integrates more naturally with RDF technologies, while still retaining the semantic information essential to entailment recognition.

In the EDS example in Figure 1, each line depicts a graph node (each corresponding to one elementary predication in the original MRS), with node identifiers prefixed to the node labels (separated by the colon), and a set of outgoing arcs (role-argument pairs) enclosed in parentheses. The semantic arguments to the relation represented by the node are directed arcs to other nodes in the EDS graph. For instance, the node for `_would_v_modal` is connected to the node for `_and_c` through an arc labeled `ARG1`. The node labeled `_and_c` in turn has outgoing arcs to `_wake_v_up` and `_fret_v_about`. The two `pron` nodes do not have outgoing arcs, they are connected to the structure through incoming arcs from the verb nodes. Finally, each of the `pronoun_q` nodes is connected to a `pron` node through a `BV` ("bound variable") arc. A graphical visualization of the same graph is shown in Figure 3 (ignoring nodes and arcs shown in green there, which are added by our entailment processor).

There are two notable examples of logic-based TE systems that have used the ERG parser and MRS meaning representations: Wotzlaw and Coote (2013) present a TE system which combines the results of deep and shallow linguistic analyses into scope-resolved MRS representations. The MRS expressions are translated into another, semantically equivalent first-order logic format, which, enriched with background knowledge, is used for the actual inference. The system of Bergmair (2010) also uses MRS as an intermediate format in constructing meaning representations. Input sentences are parsed with the ERG, and the resulting MRSs are translated into logical formulae that can be prosessed by an inference engine. In contrast to these prior applications of generic semantic parsing using the ERG to the TE task, our work simplifies the scopally underspecified logical forms of MRS into more compact graph-structured representations of core predicate–argument relations, and we define TE-specialized notions of inference over these semantic graphs.

3 Semantic Technologies and Textual Entailment

Kouylekov and Oepen (2014) map different types of meaning representations, including the EDSs used in our work, to RDF graphs, stored in off-the-shelf RDF triple stores, and searched using SPARQL queries. In our work, we build a TE system that utilizes their infrastructure as a basis for reasoning over EDS graphs.

$\{\ e_3$
$\quad x_5$:pron
$\quad \_{1}$:pronoun_q(BV x_5)
$\quad e_3$:_would_v_modal(ARG1 e_{13})
$\quad e_{11}$:_wake_v_up(ARG1 x_5)
$\quad e_{13}$:_and_c(L-INDEX e_{11}, R-INDEX e_{15}, L-HNDL e_{11}, R-HNDL e_{15})
$\quad e_{15}$:_fret_v_about(ARG1 x_5, ARG2 x_{16})
$\quad x_{16}$:pron
$\quad \_{2}$:pronoun_q(BV x_{16})
$\}$

Figure 1: EDS for *He would wake up [...] and fret about it.* (PETE id 5019).

Textual Entailment was defined by Dagan et al. (2006) as the task of recognizing whether, given two text fragments, the meaning of one text entails the meaning of the other text. The text fragments are conventionally referred to as the *text T* and the *hypothesis H*, respectively. The notion of "entailment" used in TE is informal and based at least in part on general human knowledge of language and the world.

Our textual entailment system uses graph alignment over EDS structures as the basis for entailment decisions. We extend the approach by enriching the graphs in a forward-chaining spirit using SWRL rules, and the Jena reasoner[1]. After the reasoning step, the actual alignment is performed with a SPARQL query that tries to match the hypothesis graph to the text graph. Along with a classification decision, the system outputs a "proof" by listing every SWRL rule that was used in the reasoning. In a sense, we are following the classical reasoning approach of trying to infer the hypothesis from the text.

3.1 SWRL

Our subsumption approach to entailment recognition requires some rewriting of the EDS graphs produced by the ERG parser. For example, the EDS graph in Figure 1 needs to be rewritten so that dependencies are propagated into the coordinate structure, which will facilitate the subsumption of subgraphs. We use SWRL, a semantic web standard for reasoning over ontologies[2], to encode rewriting rules for EDS graphs. The graph structures are enriched with a set of forward-chaining SWRL rules, and, thus, our graph-rewriting approach can be seen as a form of forward-chaining inference.

The system uses two sets of SWRL rules, one for the text and one for the hypothesis graph. The function of these rules is to further normalize and to add information to both graphs in order to make matching possible. We adapt the rule sets for different data sets to accomodate variation in entailment phenomena. The rule sets contain five types of rules:

- abstraction rules
- predicate simplification rules
- structural rules
- lexical relation rules
- polarity marking rules

Abstraction Rules We employ a number of abstraction rules to allow matching of indefinite and personal *pronouns* in the *H* graph to NPs in the *T* graph. To be able to match the indefinite pronoun *somebody* to the personal pronoun *he* in e.g. *He has a point he wants to make [...] ⇒ Somebody wants to make a point* (PETE id 1026), the rules label both pronouns with the same abstraction label, i.e., they add an additional `rdf:type` property to these nodes, which can be used in subsequent testing for node equivalence.

Our rules also abstract over certain *quantifiers*. In the data sets we have examined, the text and hypothesis sentence of an entailment pair often have quantifier variations that are clearly not relevant for recognizing the entailment relationship (e.g., *A woman is cleaning a shrimp ⇒ The woman is cleaning a shrimp*, SemEval id 3364). We group these quantifiers into candidate equivalence classes using rules of the form:

[1]`https://jena.apache.org/`
[2]`http://www.w3.org/Submission/SWRL/`

```
[(?a eds:predicate "_a_q") ->
(?a rdf:type eds:equiv_quant)]

[(?a eds:predicate "_the_q") ->
(?a rdf:type eds:equiv_quant)]
```

These rules state that if a node `?a` is labeled with a certain quantifier predicate (`_a_q` or `_the_q`, in this specific example), then the node `?a` is of type `equiv_quant`. This fact is added to the EDS graph, which allows matching of the node with other nodes that have the same type.

Simplified Predicates ERG lexical predicate symbols conjoin information about the lemma, part-of-speech, and sense of the wordform. To increase the robustness of the matching, we add a simplified predicate symbol which contains only the lemma and part-of-speech. This makes matching possible in cases where the ERG has given different predicate symbol interpretations of the same word in text and hypothesis. For instance, `_trade_v_in` and `_trade_v_1` are associated with different usages of the verb *trade*, and for our purposes can be simplified to `_trade_v`.

Structural Rules Certain rules enrich the graph structure without adding new meaning content to the graph. By adding arcs to certain constructions in the text graph, we make matching possible for cases where the hypothesis graph contains a substructure of the text construction. For instance, to make matching possible for the text *He would wake up [...] and fret about it* and the hypothesis *He would wake up* (PETE id 5019), we need to draw additional arcs from the node `_would_v_modal` to its indirect arguments `_wake_v_up` and `_fret_v_about`, i.e., the arguments of the conjunction node `_and_c`. This is done by applying the rules in Figure 2. The first two rules label all modal verb nodes as having type `modal_verb`, and coordinating nodes as being of type `coordination`. The third rule states that if a node is of type `modal_verb`, and it has an `ARG1` arc to a node of type `coordination`, then we add `ARG1` arcs to each of the argument nodes of the coordination. When applied to the EDS in Figure 1, the rules yield the structure shown in Figure 3, where the new arcs are marked in green.

Additional rules for **lexical relations** and **polarity marking** are described in Sections 3.3 and 3.4, respectively.

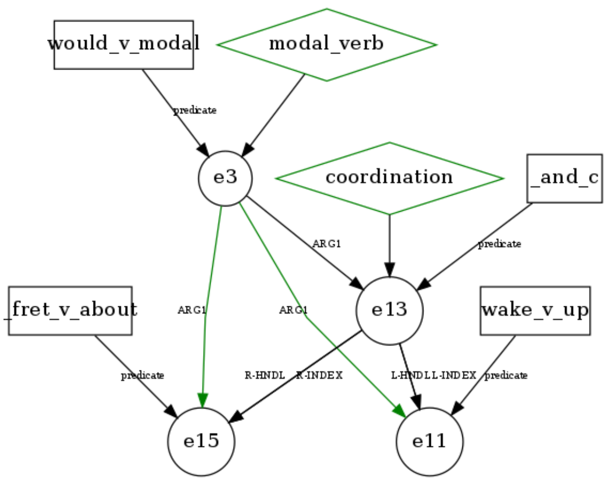


Figure 3: Additional `ARG1` arcs (in green) have been added to directly connect the modal verb (node `e3`) to its subarguments (nodes `e11` and `e15`).

Removing Graph Components To make matching possible, we also need an additional set of rules that remove certain predicates, nodes, and arcs from the hypothesis before the subsumption algorithm is applied. For instance, the sentences *A boy is playing* and *There is a boy playing* have the same meaning content, but receive different analyses from the ERG, where the existential assertion, including its tense and aspect, is reified as a separate relation. Removing the subgraph corresponding to *there is* makes matching possible. Removing graph components is not part of the general SWRL specification, but it is an extension to the rule language provided by the Jena reasoner. To ensure that the bulk of our entailment rules remain SWRL compatible, we keep the removal rules separated.

3.2 Subsumption Algorithm

Our reasoning-based system (**RBS**) processes an entailment pair using an algorithm that has the following steps:

- The text T and hypothesis H are analyzed with the ERG parser.
- The EDSs for T and H are converted into RDF triples.
- H is enriched using the SWRL rules and converted into a SPARQL query $query_h$ in which the query statements are the conjunc-

```
[modal_verb_type: (?a eds:predicate ?p), regex(?p, "^.+modal$")
-> (?a rdf:type eds:modal_verb)]

[coordination_type: (?a eds:predicate ?p), regex(?p, "^.+_c_?.*$")
-> (?a rdf:type eds:coordination)]

[(?a rdf:type eds:modal_verb),  (?b rdf:type eds:coordination),
 (?a eds:arg1 ?b), (?b eds:l-index ?c), (?b eds:r-index ?d)
-> (?a eds:arg1 ?c), (?a eds:arg1 ?d)]
```

Figure 2: SWRL rule for making explicit (inserting) arcs from a modal verb to its indirect arguments

```
_1:_a_q[BV x6]
x6:_man_n_1[]
e3:_walk_v_1[ARG1 x6]

select x1 x2 x3 where{
x1 eds:predicate "_a_q" .
x2 eds:predicate "_man_n_1" .
x3 eds:predicate "_walk_v_1" .
x1 eds:BV x2 .
x3 eds:ARG1 x2
}
```

Figure 4: Converting an EDS structure into a SPARQL query

tion of all of the triples in the RDF representation of *H*.

- The RDF triples of *T* and the SWRL rules for expanding *T* are given as the input to the reasoner.
- If the $query_h$ is matched into the inferred model for *T*, the entailment relation is assigned to the pair.

The algorithm defines textual entailment as a subsumption problem. *T* entails *H* if the (enriched) RDF graph that represents *T* contains the entire graph of *H*.

Converting *H* into a SPARQL query allows us to use the standard RDF technology to perform the graph subsumption. In Figure 4, we see an example of how the EDS for the sentence *A man walks* is converted into a SPARQL query. Using SPARQL automatically computes (and makes available) the correspondence between the predications of *T* and *H*.

Using the RDF reasoner allows us to understand the reason *H* was subsumed by *T* as the Jena reasoner outputs a verbose log on each inference step taken to obtain a specific triple in the inferred model. In the log output example below, the predication $x5$ was recognized to be an indefinite pronoun because it has the predicate `person`, and is the target of a `BV` (bound variable) relation from a predication with the predicate `_some_q`:

```
Added statement
  [x5, indef_pronoun, "true"]
Used rule
  [Rule someone-body_is_indef_pron
    concluded
    (x5 indef_pronoun 'true')
  <-
  Fact (x5 predicate 'person')
  Fact (_1 predicate '_some_q')
  Fact (_1 bv x5)]
```

3.3 Lexical Relations

In our reasoning-based system we have integrated lexical entailment rules extracted from WordNet (Fellbaum, 1998) as proposed in Lien and Kouylekov (2014). For each predication in *T* we dynamically create SWRL rules that expand the RDF graph of *T* by adding new predications for words that are synonyms or hypernyms of the original predication. For example, for the predication `_assistant_n` we expand the *T* graph with the predications `_worker_n` and `_person_n`. Figure 5 shows a simplified version of these rules.

The creation of these rules is done once before the start of the inference. The system queries WordNet for rules that can be used until no rules can be added. If the SWRL rules add predicates after the reasoning step that can be expanded using rules deducted from WordNet then these rules are added to the reasoner and the reasoning is restarted. We used this strategy as we were not able to encode the entire WordNet database as rules in

```
[to-sense-rule: (x eds:predicate "_assistant_n_1")
      -> (x eds:wordnet "assistant_n_1") ]

[hypernym-rule: (x eds:wordnet "assistant_n_1")
      -> (x eds:wordnet "worker_n_1")]

[hypernym-rule: (x eds:wordnet "worker_n_1")
      -> (x eds:wordnet "person_n_1")]

[to-predicate-rule: (x eds:wordnet "person_n_1")
      -> (x eds:predicate "_person_n_1")]
```

Figure 5: Automatically generated WordNet rules.

the Jena reasoner.

3.4 Contradiction

The SemEval 2014 task uses a three-way classification of the entailment pairs. Systems were required to assign to each pair one of the three categories ENTAILMENT, CONTRADICTION, or NEUTRAL. To handle three-way classification, we have developed a special rule-based contradiction module. Although the SemEval data display various contradiction phenomena, we focus on negation, which is the most frequent contradiction indicator.

For classification of pairs where event negation or instance negation in one of the sentences creates contradiction, we combine *polarity marking* of nodes with graph matching. The nodes that are in the immediate scope of the negation are marked as negative, and all other nodes as positive. For instance, in the most simple case of event negation, the predicate `neg` negates some event node via an `ARG1` arc (e.g., *not singing*). The following rule marks both the node of the `neg` predicate, and the event node as negative:

```
[(?a eds:predicate "neg"),
 (?a eds:arg1 ?b) ->
 (?a eds:polarity "negative"),
 (?b eds:polarity "negative")]
```

In the parallell case for simple instance negation (e.g., *no woman*), the node of the `_no_q` predicate and its "bound variable", the instance node, are both labeled as negative:

```
[(?q eds:predicate "_no_q"),
 (?q eds:bv ?a) ->
 (?q eds:polarity "negative"),
 (?a eds:polarity "negative")]
```

Since both events and instances can be complex linguistic constructions, our rule set contains rules that handle negation of e.g., compounds, nominalizations, coordination, and nesting of verbs. Broadly speaking, these rules are similar in spirit to the "MRS crawling" process defined by Packard et al. (2014) for the task of negation scope resolution.

In the classification process, we run the system twice on each entailment pair: in the first run the polarity markings are ignored, and in the second run they are considered. If the system finds a subsumption of H in the T graph *without* polarity markings, but no subsumption *with* polarity markings, then the pair is classified as CONTRADICTION.

Polarity marking allows us to use the same structures for both entailment and contradiction testing. Our polarity marking approach is parallell to how negation is represented in AMR[3].

3.5 N-best Matching

The ERG parser can output a ranked list of candidate analyses for a sentence. We extended our system with n-best matching to facilitate entailment recognition when the top-ranked analysis does not correspond to the perceived meaning of the sentence, i.e., to reduce the impact of errors in parse ranking. Such errors include prepositional phrase attachments, noun compounds, coordinate structures, and other interpretation variants. For example in the sentence:

who invented the light bulb?

[3]`https://github.com/amrisi/amr-guidelines/blob/master/amr.md#negation`

the parser creates two valid (in principle, if not equally likely) analyses based on the semantic interpretation of the word *light* as 1) an adjective; 2) part of a noun–noun compound. If the same phrase occurs in *T* and *H*, but their contexts are different, the top-ranked analyses from the parser ranker for *T* and *H* may contain different interpretations of the phrase. Our default assumption is that such misalignment is the cause of many unwarranted mismatches between the *T* and *H* graphs.

For each entailment pair i ($pair_i$) we iterate over all analyses of *T* and *H*. If the n-th analysis of *T* entails the k-th analysis of *H* we assign the ENTAILMENT relation to the entailment pair. This definition is valid as each analysis of *T* and *H* corresponds to a valid interpretation.

To determine the number of analyses for *T* and *H* we need to consider [4] we have employed an optimization strategy. We have gradually increased the number of considered analyses of *T* and *H*, and measured the system performance on the training set. The best n-m combination, where n are the analyses considered for *T* and m are the analyses considered for *H*, is used on the test set.

4 First Case Study: PETE

In our first case study, we recast the Lien (2014) heuristic for the PETE shared task data as SWRL rules. The objective of the PETE task was to propose an alternative method for parser evaluation: instead of comparing parser output to gold annotated treebank data, parsers can be evaluated indirectly by examining how well the parser output supports the task of entailment recognition. The data provided for the task was constructed so that syntactic analysis of the sentence pairs would be sufficient to determine whether the text entails the hypothesis. The PETE development and test sets contain 66 and 301 sentence pairs, respectively. Characteristically, the hypothesis sentence of the positive entailment pairs is shorter that the text sentence, and is a substructure of the text, frequently with some minor changes (e.g., active-to-passive conversion, a noun phrase in the text is replaced by a underspecified pronoun in the hypothesis). In the negative entailment pairs the hypothesis usually contains elements from the text that are structured differently and thus give the hypothesis a different meaning from the text.

[4]The ERG can return all the possible grammatical analyses up to a user-supplied maximum rank n.

The best scoring system in the shared task was the Cambridge system (Rimell and Clark, 2010), with an accuracy of 72.4%.

Table 1 presents our 1-best and 10-best results on the PETE test data, and compares them to the results reported by Lien (2014), and the shared task winner Rimell and Clark (2010). Our **RBS** system outperforms the system developed by Lien (2014), establishing a new state-of-the-art. The two systems have close results on both single analysis input and n-best. This demonstrates that our system correctly implements the approach proposed in Lien (2014).

The main advantage of our system is the high **precision**. The PETE data focus on entailments that can be recognized using structural analysis alone (allowing for the substitution of noun phrases with generalized pronouns), which fits nicely with our strict graph subsumption algorithm over meaning representations. When we examine the system's output for the PETE development data, we see that two-thirds of the true positives in the ENTAILMENT category concern sentence pairs where *H* is a substructure of *T*. In these cases, enriching the RDF graphs with arcs connecting predicates to their indirect arguments, and allowing noun phrases to match generalized pronouns, is sufficient for entailment recognition. In the remaining one-third of the true positives, there are syntactic differences from *T* to *H*, but the ERG abstracts from these differences and assigns the same analysis to both (the relevant substring of) *T* and *H*. For instance, the *T* noun phrase *steamed, whole-wheat grains* and the *H* sentence *Grains are steamed* (PETE id 3081.N) receive the same EDS analysis, with *grains* as the passive `ARG2` of the verb *steam*. In another example below (PETE id 2004), the relative pronoun *which* is ignored at the level of ERG semantics, which instead directly identifies *the stream* as the `ARG2` of the seeing event:

> *[...] the stream which he had seen [...].*
> $\Rightarrow$ *Someone had seen the stream.*

In the cases where our system fails to recognize the entailment relationship, it is often the case that one of the sentences is assigned an incorrect analysis from the ERG parser. An incorrect assignment of an argument role, or an incorrect attachment of a prepositional phrase prevents our strict subsumption algorithm from classifying the relationship as entailment.

	RBS	RBS n-best	Lien	Lien n-best	Rimell & Clark
Accuracy	**72.1**	**77.1**	70.7	76.4	72.4
Precision	89.0	81.1	88.6	81.4	79.6
Recall	52.6	72.7	50.0	70.5	62.8
F-Measure	66.1	76.6	63.9	75.5	70.2

Table 1: Performance of our reasoning-based system on the PETE test data.

The influence of imperfect parse ranking on the system performance can be alleviated by running it on n-best parser outputs. Considering multiple analyses of *T* and *H* from the ERG parser increases the performance of our system by adding a significant boost to the recall without damaging the precision. Using 1-best analyses for *T* and *H*, our system has a performance compatible with the previously best performing system on the PETE task.

5 Second Case Study: SemEval 2014 Task 1

RBS	RBS n-best	UIO-Lien	Illinois-LH
77.4	80.4	77.1	84.6

Table 2: Comparison of accuracy of RBS on the SemEval test data.

	Precision	Recall	F-Measure
Contradiction	95.9	66.1	78.3
Entailment	95.6	52.4	67.7

Table 3: Precision, recall and F-measure of RBS n-best on the SemEval test data.

In our second case study, we revisit our contribution to the SemEval 2014 task 1. The focus of this task was evaluation of compositional distributional semantic models through entailment decision (and semantic relatedness) on sentence pairs, in order to remedy the lack of benchmarks for such models. The 10,000 sentence pair data set released for the task (50% training, 50% test) reflects this goal by targeting phenomena that compositional distributional semantic models are meant to account for, e.g., lexical variation phenomena such as contextual synonymy, active-passive and other syntactic alternation, negation, and operator scope. The data do not require encyclopedic knowledge about instances of concepts, only generic semantic knowledge about general concept categories. Unlike in the PETE data set, the text and hypothesis sentences are usually similar in length, and either paraphrase or contradict each other, or are more or less unrelated in meaning.

In the entailment subtask, systems were required to assign one of the categories ENTAILMENT, CONTRADICTION, or NEUTRAL to each sentence pair. The best scoring system was the Illinois-LH system (Lai and Hockenmaier, 2014), with an accuracy of 84.6%.

Table 2 presents our 1-best and 10-best results on the SemEval test data, and compares them to the results for the UIO-Lien system (Lien and Kouylekov, 2014) and the shared task winner Illinois-LH.

The results obtained on the SemEval data set are encouraging. As with the PETE data set we have improved over the results we achieved with the UIO-Lien system. This demonstrates the adaptability of our approach to new data sets. When we participated in the SemEval task with the UIO-Lien system, we did not submit a run using the n-best analyses from the ERG parser, so we are not able to make a comparison for n-best results. Our current n-best RBS system obtains a high accuracy which makes it the 6th ranked system on the SemEval data. With this result it is the top ranked unsupervised rule based system.

It is worth noticing that our system achieves a similar result as another task participant, Bestgen (2014), which employs a similarity-based algorithm and latents semantic analysis to recognize entailment. Our advantage versus such approaches is that we are able to create a reasoning chain that motivates the system decision instead of presenting a simple similarity number. Still in the future development of our system we can investigate the possibility of using probabilistic rules to guide our reasoner.

Similar to the PETE dataset results our system obtained a high precision (more than 95.0% precision on both ENTAILMENT and CONTRADIC-

TION), maintaining a decent recall as shown in Table 3.

The SemEval data display more variation in entailment phenomena than the PETE data, and require the use of external knowledge sources. We use WordNet to generate lexical inference rules. This allows us to capture the same types of "syntactic" entailments as in the PETE data, augmented with synonymy and hypernymy relations between predicates in *T* and *H*, as examplified by the following entailment pair (SemEval id 4176):

An eggplant is being sliced by a woman
⇒ A woman is cutting a vegetable

We did not focus on capturing entailment phenomena that were aimed specifically at evaluation of compositional distributional semantic models, and that require contextual information or equating structurally diverse phrases. In many cases, it would require formulating specific rules that would do little to improve the coverage of our system.

The system's high precision on ENTAILMENT shows that the graph subsumption of semantic structures is a reliable indicator of the entailment relation. To further improve recall, the system must incorporate more sources of knowledge and semantic variation.

6 Conclusions and Future Work

In this paper we have described an approach to TE which leans heavily on generic semantic parsing technologies, combining the off-the-shelf ERG parser with formats and tools developed for the Semantic Web and a custom-built notion of inference over graph-structured meaning representations. We have replicated our two previous TE shared task contributions, and using n-best analyses reached state-of-the-art for rule-based TE systems. These results demonstrate the utility of general-purpose, off-the-shelf semantic parsing systems for textual entailment, in particular when reasoning over ranked n-best lists can be applied to compensate for parse ranking limitations. Our system architecture rests on a comparatively small number of reasonably generic rules, i.e., there is very little task-specific engineering and tuning in our approach (as a large part of the work in done in the parser). Our 95 percent precision results demonstrate that subsumption of semantic representations is a strong indication for textual entailment. Our work contributes to moving the TE field towards logical reasoning.

One of the main strength of the system is its versatility. We reduce the amount of task-specific engineering by using generic off-the-shelf tools.

Future Work Our approach is useful for precision-critical applications like information retrieval and particularly Question Answering. In future work we plan to combine it with a shallow information retrieval approach and use its evaluation power to pick the correct answer. The system also provides a detailed account of the reasoning behind each entailment decision. This strength can be used in an answer presentation module which motivates why the system has chosen a particular answer.

References

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract meaning representation for sembanking. In *Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse*, page 178 – 186, Sofia, Bulgaria, August.

Roy Bar-Haim, Ido Dagan, Iddo Greental, and Eyal Shnarch. 2007. Semantic inference and the lexical-syntactic level. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 871–876.

Richard Bergmair. 2010. *Monte Carlo Semantics: Robust Inference and Logical Pattern Processing with Natural Language Text*. Ph.D. thesis, University of Cambridge.

Yves Bestgen. 2014. CECL: a new baseline and a noncompositional approach for the sick benchmark. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 1–8, Dublin, Ireland, August. Association for Computational Linguistics and Dublin City University.

Johan Bos and Katja Markert. 2006. When logical inference helps determining textual entailment (and when it doesn't). In Bernardo Magnini and Ido Dagan, editors, *The Second PASCAL Recognising Textual Entailment Challenge. Proceedings of the Challenges Workshop*, pages 98–103, Venice, Italy.

Ulrich Callmeier. 2002. Preprocessing and encoding techniques in PET. In Stephan Oepen, Daniel Flickinger, J. Tsujii, and Hans Uszkoreit, editors, *Collaborative Language Engineering. A Case Study in Efficient Grammar-based Processing*, page 127 – 140. CSLI Publications, Stanford, CA.

Ann Copestake, Dan Flickinger, Carl Pollard, and Ivan A. Sag. 2005. Minimal Recursion Semantics. An introduction. *Research on Language and Computation*, 3(4):281 – 332.

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2006. The PASCAL recognising textual entailment challenge. In Joaquin Quiñonero-Candela, Ido Dagan, Bernardo Magnini, and Florence d'Alché Buc, editors, *Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Tectual Entailment*, volume 3944 of *Lecture Notes in Computer Science*, page 177 – 190. Springer Berlin Heidelberg.

Christiane Fellbaum. 1998. *WordNet: An Electronic Lexical Database*. Bradford Books.

Dan Flickinger. 2000. On building a more efficient grammar by exploiting types. *Natural Language Engineering*, 6 (1):15 – 28.

Milen Kouylekov and Stephan Oepen. 2014. RDF Triple Stores and a Custom SPARQL Front-End for Indexing and Searching (Very) Large Semantic Networks. In *COLING 2014, 25th International Conference on Computational Linguistics, Proceedings of the Conference System Demonstrations, August 23-29, 2014, Dublin, Ireland*, pages 90–94.

Alice Lai and Julia Hockenmaier. 2014. Illinois-LH: A denotational and distributional approach to semantics. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 329–334, Dublin, Ireland, August. Association for Computational Linguistics and Dublin City University.

Elisabeth Lien and Milen Kouylekov. 2014. UIO-Lien: Entailment recognition using minimal recursion semantics. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 699–703, Dublin, Ireland, August. Association for Computational Linguistics and Dublin City University.

Elisabeth Lien. 2014. Using minimal recursion semantics for entailment recognition. In *Proceedings of the Student Research Workshop at the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 76–84, Gothenburg, Sweden, April. Association for Computational Linguistics.

Marco Marelli, Luisa Bentivogli, Marco Baroni, Raffaella Bernardi, Stefano Menini, and Roberto Zamparelli. 2014. Semeval-2014 task 1: Evaluation of compositional distributional semantic models on full sentences through semantic relatedness and textual entailment. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 1–8, Dublin, Ireland, August. Association for Computational Linguistics and Dublin City University.

Stephan Oepen and Jan Tore Lønning. 2006. Discriminant-based MRS banking. In *Proceedings of the 5th International Conference on Language Resources and Evaluation*, page 1250 – 1255, Genoa, Italy.

Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Dan Flickinger, Jan Hajič, Angelina Ivanova, and Yi Zhang. 2014. SemEval 2014 Task 8. Broad-coverage semantic dependency parsing. In *Proceedings of the 8th International Workshop on Semantic Evaluation*, Dublin, Ireland.

Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Silvie Cinkova, Dan Flickinger, Jan Hajic, and Zdenka Uresova. 2015. Semeval 2015 task 18: Broad-coverage semantic dependency parsing. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 915–926, Denver, Colorado, June. Association for Computational Linguistics.

Woodley Packard, Emily M. Bender, Jonathon Read, Stephan Oepen, and Rebecca Dridan. 2014. Simple negation scope resolution through deep parsing: A semantic solution to a semantic problem. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 69–78, Baltimore, Maryland, June. Association for Computational Linguistics.

Laura Rimell and Stephen Clark. 2010. Cambridge: Parser Evaluation using Textual Entailment by Grammatical Relation Comparison. In *Proceedings of the 5th International Workshop on Semantic Evaluation, ACL 2010*.

Marta Tatu and Dan Moldovan. 2006. A logic-based semantic approach to recognizing textual entailment. In *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, pages 819–826, Sydney, Australia, July. Association for Computational Linguistics.

Andreas Wotzlaw and Ravi Coote. 2013. A Logic-based Approach for Recognizing Textual Entailment Supported by Ontological Background Knowledge. *CoRR*, abs/1310.4938.

Deniz Yuret, Aydin Han, and Zehra Turgut. 2010. SemEval-2010 Task 12: Parser Evaluation using Textual Entailments. In *Proceedings of the 5th International Workshop on Semantic Evaluation*, pages 51–56. Association for Computational Linguistics.

A Framework for Procedural Text Understanding

Hirokuni Maeta[*]
Cybozu, Inc.
2-7-1 Nihombashi, Chuo-ku,
Tokyo, Japan
hirokuni.maeta@gmail.com

Tetsuro Sasada
ACCMS, Kyoto University
Yoshida Honmachi, Sakyo-ku,
Kyoto, Japan
sasada@ar.media.kyoto-u.ac.jp

Shinsuke Mori
ACCMS, Kyoto University
Yoshida Honmachi, Sakyo-ku,
Kyoto, Japan
forest@i.kyoto-u.ac.jp

Abstract

In this paper we propose a framework for procedural text understanding. Procedural texts are relatively clear without modality nor dependence on viewpoints, etc. and have many potential applications in artificial intelligence. Thus they are suitable as the first target of natural language understanding. As our framework we extend parsing technologies to connect important concepts in a text. Our framework first tokenizes the input text, a sequence of sentences, then recognizes important concepts like named entity recognition, and finally connect them like a sentence parser but dealing all the concepts in the text at once. We tested our framework on cooking recipe texts annotated with a directed acyclic graph as their meaning. We present experimental results and evaluate our framework.

1 Introduction

Among many sorts of texts in natural languages, procedural texts are clear and related to the real world. Thus they are suitable for the first target of natural language understanding (NLU). A procedural text is a sequence of sentences describing instructions to create an object or to change an object into a certain state. If a computer understands procedural texts, there are potentially tremendous applications: an intelligent search engine for how-to texts (Wang et al., 2008), more intelligent computer vision (Ramanathan et al., 2013), a work help system teaching the operator what to do the next (Hashimoto et al., 2008), etc.

The general natural language processing (NLP) tries to solve the understanding problem by a long series of sub-problems: word identification, part-of-speech tagging, parsing, semantic analysis, and so on. Contrary to this design, in this paper, we propose a concise framework of NLU focusing on procedural texts. There have been a few attempts at procedural text understanding. Momouchi (1980) tried to convert various procedural texts into so-called PT-chart on the background of automatic programming. Hamada et al. (2000) proposed a method for interpreting cooking instruction texts (recipes) to schedule two or more recipes. Although their definition of understanding was not clear and their approach was based on domain specific heuristic rules, these pioneer works inspired us to tackle a major problem of NLP, text understanding.

As the meaning representation of a procedural text we adopt a flow graph. Its vertices are important concepts consisting of word sequences denoting materials, tools, actions, etc. And its arcs denote relationships among them. It has a special vertex, root, corresponding to the final product. The problem which we try to solve in this paper is to convert a procedural text into the appropriate flow graph. The input of our NLU system is the entire text, but not a single sentence.

Our framework first segments sentences into words (word segmentation; abbreviated to WS hereafter). This process is only needed for some languages without clear word boundary. Then we identify concepts in the texts and classify them into some categories (concept identification; abbreviated to CI hereafter). And finally we connect them with labeled arcs. For the first process, WS, we adapt an existing tool to the target domain and achieve an enough high accuracy. The second process, CI, can be solved by the named entity recognition (NER) technique given an annotated corpus (training data). The major difference is the definition of named entities (NE). Contrary to many other NERs we propose a method that does not

[*]This work was done when the first author was at Kyoto University.

require part-of-speech (POS) tags. This makes our text understanding framework simple. For the final process we extend a graph-based parsing method to deal with the entire text, a sequence of sentences, at once. The difference from sentence parsing is that the vertices are concepts but not words and there are words not covered by any concept functioning as clues for the structure.

As a representative of procedural texts, we selected cooking recipes, because there are many available resources not only in the NLP area but in the computer vision (CV) area. For example, the TACoS dataset (Regneri et al., 2013), is a collection of short videos recording fundamental actions in cooking with descriptions written by Amazon Mechanical Turk. Another example, the KUSK dataset (Hashimoto et al., 2014), contains 40 videos recording entire executions (20 recipes by two persons). The recipes in the KUSK dataset are taken from the r-FG corpus (Mori et al., 2014), in which each recipe text is annotated with its "meaning."

We tested our framework on recipe texts manually annotated with word boundary information, concepts, and a flow graph. We compare a naive application of an MST dependency parser and our extension for flow graph estimation. We also measure the accuracy at each step with the gold input assuming the perfect preceding steps. Finally we evaluate the full automatic process of building a flow graph from a raw text. Our result can be a solid baseline for future improvement in the procedural text understanding problem.

2 Related Work

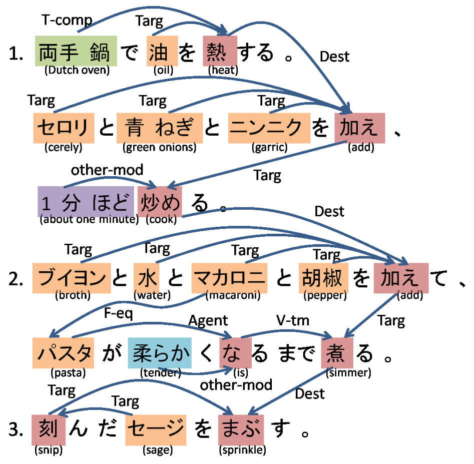
Some attempts at procedural text understanding were proposed in the early 80's (Momouchi, 1980). Then Hamada et al. (2000) proposed tree-based representation of cooking instruction texts (recipes) from the application point of view. These approaches used rule-based methods, but they, along with the current success of the machine learning approach, inspired us to conceive that the procedural text understanding can be a tractable problem for the current NLP.

In our framework the procedural text understanding problem is decomposed into three processes. The first process is the well-known WS. There have been many researches reporting high accuracies in various languages based on the corpus-based approach (Merialdo, 1994; Neubig et al., 2011, *inter alia*). The second one is CI, which can be solved in the same way of NER (Chinchor, 1998) with a different definition of named entities. The accuracy of the general NER is less than WS but is more than 90% when a large annotated corpus is available (Sang and Meulder, 2003, *inter alia*). So we can say that CI can also be solved given an annotated corpus. The only open question is how many examples are required to achieve a practically high accuracy. This paper gives a solution to this. The third one is our original text parsing, which outputs a flow graph taking a text and the concepts in it as the input. To solve this problem, we follow the idea of the graph-based dependency parsing (McDonald et al., 2006; McDonald et al., 2005). Dependency parsing attempts to connect *all the words* in an input sentence with labeled arcs to form a rooted tree. In our method, the units are concepts instead of words and the input is an entire text (a sequence of sentences), not a single sentence. The words not forming concepts (mainly function words), are only referred to as features to estimate the flow graph. We also add another module to form a directed acyclic graph (DAG).

From the NLP viewpoint, the major problems we are solving are 1) dependency parsing (Buchholz and Marsi, 2006) among concepts only, 2) predicate-argument structure analysis (Taira et al., 2010; Yoshino et al., 2013), 3) semantic parsing (Wong and Mooney, 2007; Zettlemoyer and Collins, 2005), and 4) coreference, anaphora, and ellipsis resolution (Nielsen, 2004; Fernández et al., 2004). For dependency parsing we resolve the target of modifiers such as quantities, durations, timing clauses. For predicate-argument structure analysis, we figure out which action is applied to what object with what tools, even if it is stated in passive form or just by a past participle. For semantic parsing we resolve the relationships between concepts. For coreference, anaphora, and ellipsis resolution, our DAG constructor links an action to another action that takes the result of the former action or an abstract expression to a concrete intermediate product. Our method solves these problems focusing on important notions at once.

The understanding of procedural texts may allow a more sophisticated combination of NLP an CV. Recently there have been some attempts at aligning videos and natural language descriptions

1. 両手 鍋$_{\mathrm{T}}$ で 油$_{\mathrm{F}}$ を 熱$_{\mathrm{Ac}}$ する 。
   (In a Dutch oven$_{\mathrm{T}}$, heat$_{\mathrm{Ac}}$ oil$_{\mathrm{F}}$.)
   セロリ$_{\mathrm{F}}$ と 緑 玉ねぎ$_{\mathrm{F}}$ と ニンニク$_{\mathrm{F}}$ を 加え$_{\mathrm{Ac}}$ る 。
   (Add$_{\mathrm{Ac}}$ celery$_{\mathrm{F}}$, green onions$_{\mathrm{F}}$, and garlic$_{\mathrm{F}}$.)
   1 分 ほど$_{\mathrm{D}}$ 炒め$_{\mathrm{Ac}}$ る 。
   (Cook$_{\mathrm{Ac}}$ for about 1 minute$_{\mathrm{D}}$.)
2. ブイヨン$_{\mathrm{F}}$ と 水$_{\mathrm{F}}$ と マカロニ$_{\mathrm{F}}$ と 胡椒$_{\mathrm{F}}$ を 加え$_{\mathrm{Ac}}$ て 、 パスタ$_{\mathrm{F}}$ が 柔らか$_{\mathrm{Sf}}$ く な$_{\mathrm{Af}}$ る まで 煮$_{\mathrm{Ac}}$ る 。
   (Add$_{\mathrm{Ac}}$ broth$_{\mathrm{F}}$, the water$_{\mathrm{F}}$, macaroni$_{\mathrm{F}}$, and pepper$_{\mathrm{F}}$, and simmer$_{\mathrm{Ac}}$ until the pasta$_{\mathrm{F}}$ is$_{\mathrm{Af}}$ tender$_{\mathrm{Sf}}$.)
3. 刻$_{\mathrm{Ac}}$ ん だ セージ$_{\mathrm{F}}$ を まぶ$_{\mathrm{Ac}}$ す 。
   (Sprinkle$_{\mathrm{Ac}}$ the snipped$_{\mathrm{Ac}}$ sage$_{\mathrm{F}}$.)



T-comp
Targ
1. 両手 鍋 で 油 を 熱 する 。
(Dutch oven) (oil) (heat)
Dest
Targ
Targ
Targ
セロリ と 青 ねぎ と ニンニク を 加え 、
(cerely) (green onions) (garric) (add)
other-mod
Targ
1 分 ほど 炒め る 。
(about one minute) (cook)
Dest
Targ
Targ
Targ
Targ
2. ブイヨン と 水 と マカロニ と 胡椒 を 加え て 、
(broth) (water) (macaroni) (pepper) (add)
F-eq
Agent
V-tm
Targ
パスタ が 柔らか く な る まで 煮 る 。
(pasta) (tender) (is) (simmer)
Targ
other-mod
Dest
Targ
3. 刻 ん だ セージ を まぶ す 。
(snip) (sage) (sprinkle)

Figure 1: Examples of a procedural text and its flow graph.

(Naim et al., 2014; Rohrbach et al., 2013). In these researches, the NLP part is very naive. They just identify the nouns in the text and apply a sequence-based alignment tool. Now the machine translation community is shifting to the tree-based approach to capture structural differences in two languages. The flow graph representation enables grounding of tuples consisting of an action and its target objects, and also absorbs the difference in the execution order of a procedural text and the video recording its execution.

Although NLU is the major scientific problem of AI, procedural text understanding is important from the viewpoint of applications as well. For cooking recipes for example, on which we test our framework in this paper, we can realize a more intelligent search engine, summarization, or a help system (Wang et al., 2008; Yamakata et al., 2013; Hashimoto et al., 2008).

3 Recipe Flow Graph Corpus

As a test bed of the text parsing problem, we adopt the recipe flow graph corpus (r-FG corpus) (Mori et al., 2014). To our best knowledge, this is the only corpus annotated with flow graphs that matches with our requirements. In addition cooking recipes are representative procedural texts describing very familiar activities of our daily life, and its meaning representation has various applications. Our framework is, however, not limited to this corpus.

3.1 r-FG Corpus

The r-FG corpus contains randomly crawled recipes in Japanese from a famous Internet recipe

#recipes	#sentences	#NEs	#words
200	1,303	7,268	25,446

Table 1: Corpus statistics.

site.[1] The specification of the corpus is shown in Table 1. The text part of a recipe consists of a sequence of steps and the steps have some sentences. All the concepts (entities and actions) appearing in the sentences are identified and annotated with a concept tag.[2] The text part is annotated with a rooted DAG representing its meaning as shown in Figure 1.

3.2 Vertices

Each vertex of a flow graph corresponds to a concept represented by a word sequence in the text and a concept type such as food, tool, action. Table 2 lists the concept types along with the average number of occurrences per recipe. There is one special vertex, root, corresponding to the final dish. In the Figure 1 example, the node of "splinkle" is the root.

3.3 Arcs

An arc between two vertices indicates that they have a certain relationship. An arc has a label denoting its relationship type. Table 3 lists the arc types with their average frequencies per recipe. The most interesting relationships may be coreferences and null-instantiated arguments. In Fig-

[1]`http://cookpad.com` (accessed on 2015 May 19)

[2]In the original r-FG paper (Mori et al., 2014), they call the concepts "recipe named entities." In this paper we use the term "concept" to refer to them, because the recipe named entities contain verb phrases.

Concept tag	Meaning	Freq.
F	Food	11.87
T	Tool	3.83
D	Duration	0.67
Q	Quantity	0.79
Ac	Action by the chef	13.83
Af	Action by foods	2.04
Sf	State of foods	3.02
St	State of tools	0.30
Total	–	36.34

Table 2: Concept tags with frequencies per recipe.

Arc label	Meaning	Freq.
Agent	Action agent	2.15
Targ	Action target	15.67
Dest	Action destination	7.22
F-comp	Food complement	0.65
T-comp	Tool complement	1.32
F-eq	Food equality	3.15
F-part-of	Food part-of	2.37
F-set	Food set	0.15
T-eq	Tool equality	0.44
T-part-of	Tool part-of	0.39
A-eq	Action equality	0.53
V-tm	Head of a clause for timing	1.06
other-mod	Other relationships	3.54
Total	–	38.62

Table 3: Arc labels with frequencies per recipe.

ure 1 for example, "macaroni" is equal to "pasta." According to the world knowledge, macaroni is a sort of pasta, but in this recipe they are identical. An example of a null-instantiated argument is the relationship between "heat" and "add." Celery etc. should be added not to the initial cold Dutch oven without oil but to the hot Dutch oven with oil, which is the implicit result of the action "heat."

4 Overview of Procedural Text Understanding

Our framework of procedural text understanding consists of the following three processes combined in the cascaded manner.

1. Word segmentation (WS)
2. Concept identification (CI)
3. Flow graph estimation

The input of WS is a raw sentence and the output is a word sequence. For example the WS takes the first sentence in Figure 1 without any tag as the input as follows:

両手鍋で油を熱する。

Then WS outputs the following word sequence separated by whitespace as the output.

両手 鍋 で 油 を 熱 する 。

The input of CI is the word sequence, the output of WS, and it identifies concepts, which are spans of words without overlap annotated with its type sequences. For the above example, the CI outputs three concepts as follows:

両手 鍋$_{\mathsf{F}}$ で 油$_{\mathsf{F}}$ を 熱$_{\mathsf{Ac}}$ する 。

This part is similar to NER. Contrary to a normal NER, however, our method does not require POS tag for the words in the input. Thus we do not need to adapt a POS tagger to the target domain. For English or other languages with obvious word boundary, we can start from CI.

Now we have a text consisting of some sentences with concepts identified. An example is the left hand side of Figure 1. This is the input of the flow graph estimation step and the output is a flow graph as show on the right hand side of Figure 1 for example.

In the traditional NLP approach, many sub-problems proceed after NER. Syntactic parsing clarifies the intra-sentential relationships among NEs, then anaphora/coreference resolution figures out their inter-sentential relationships. Contrary, we process the entire text at once. In the subsequent section, we describe the above three process in detail.

5 Word Segmentation

Some languages such as Japanese or Chinese, have no obvious word boundary like whitespace in English. The first step of our framework is WS. For many European languages this process is almost obvious and instead of WS we only need to decompose some special words like "isn't" to "is" + "not" in English or "du" to "de" + "le" in French.

For WS we adopt the pointwise method (Neubig et al., 2011) because of its flexibility for language resource addition.[3] This characteristics is suitable especially for domain adaptation. Below we explain pointwise WS briefly and our method to improve its accuracy for user generated recipes.

[3]An implementation and the default model for the general domain are available from `http://www.phontron.com/kytea/` (accessed on 2015 May 19).

Type	Feature setting
Character n-gram	x_{i-2}, x_{i-1}, x_i, x_{i+1}, x_{i+2}, x_{i+3}, $x_{i-2}x_{i-1}$, $x_{i-1}x_i$, x_ix_{i+1}, $x_{i+1}x_{i+2}$, $x_{i+2}x_{i+3}$, $x_{i-2}x_{i-1}x_i$, $x_{i-1}x_ix_{i+1}$, $x_ix_{i+1}x_{i+2}$, $x_{i+1}x_{i+2}x_{i+3}$
Character type n-gram	$c(x_{i-2})$, $c(x_{i-1})$, $c(x_i)$, $c(x_{i+1})$, $c(x_{i+2})$, $c(x_{i+3})$, $c(x_{i-2})c(x_{i-1})$, $c(x_{i-1})c(x_i)$, $c(x_i)c(x_{i+1})$, $c(x_{i+1})c(x_{i+2})$, $c(x_{i+2})c(x_{i+3})$, $c(x_{i-2})c(x_{i-1})c(x_i)$, $c(x_{i-1})c(x_i)c(x_{i+1})$ $c(x_i)c(x_{i+1})c(x_{i+2})$, $c(x_{i+1})c(x_{i+2})c(x_{i+3})$
Dictionary	$d(x_{i-2}x_{i-1}x_ix_{i+1})$, $d(x_{i-1}x_ix_{i+1}x_{i+2})$, $d(x_ix_{i+1}x_{i+2}x_{i+3})$ $L(\cdots x_{i-2}x_{i-1}x_i)$, $R(x_{i+1}x_{i+2}x_{i+3}\cdots)$

Table 4: Features for word segmentation. The fuction $c(\cdot)$ maps a character into one of six character types: symbol, alphabet, arabic, number *hiragana*, *katakana*, and *kanji*. The fuction $d(\cdot)$ returns whether the string is in the dictionary or not. And the functions $L(\cdot)$ and $R(\cdot)$ return whether substrings of any length on the left hand side or right hand side match with a dictionary entry.

5.1 Pointwise Method

The pointwise method formulate WS as a binary classification problem, estimating boundary tags $\boldsymbol{b}_1^{I-1}$. Tag $b_i = 1$ indicates that a word boundary exists between characters x_i and x_{i+1}, while $b_i = 0$ indicates that a word boundary does not exist. This classification problem can be solved by tools in the standard machine learning toolbox such as support vector machines (SVMs).

The features are character n-grams surrounding the decision point i, which are substrings of $x_{i-2}x_{i-1}x_ix_{i+1}x_{i+2}x_{i+3}$, character type n-grams, and whether character n-grams matches an entry in the dictionary or not. Table 4 lists the features.

As we can see, the pointwise WS does not refer to the other decisions, thus we can train it from a partially segmented sentences, in which only some points between characters are annotated with word boundary information.

5.2 Domain Adaptation

As the WS adaptation to recipes, we convert the r-FG corpus into partially segmented sentences following (Mori and Neubig, 2014). In the corpus only r-NEs are segmented into words. That is to say, only both edges of the r-NEs and the inside of the r-NEs are annotated with word boundary information. If the r-NE in focus is ホットドッグ composed of two words, then the partially segmented sentences are

ex.) 各|ホ–ッ–ト|ド–ッ–グ|に␣チ␣リ␣、…,

ex.) |ホ–ッ–ト|ド–ッ–グ|を␣ア␣ル␣ミ…,

where the symbols "|," "–," and "␣" mean word boundary, no word boundary, and no information, respectively. Then we use the partially annotated sentences which we obtained in this way as an additional language resource to train the model.

Type	Feature setting
Word n-gram	w_{i-2}, w_{i-1}, w_i, w_{i+1}, w_{i+2}, $w_{i-2}w_{i-1}$, $w_{i-1}w_i$, w_iw_{i+1}, $w_{i+1}w_{i+2}$, $w_{i-2}w_{i-1}w_i$, $w_{i-1}w_iw_{i+1}$, $w_iw_{i+1}w_{i+2}$

Table 5: Features for concept identification.

6 Concept Identification

The second step is the concept identification. The concept in the text parsing problem is a span of words without overlap annotated with its type. Thus the concept identification (CI) can be solved in the same manner as the named entity recognition (NER). NER is a sequence labeling problem and many solutions have been proposed so far (Borthwick, 1999; Sang and Meulder, 2003, *inter alia*).

The standard NER method is based on linear chain conditional random fields (CRFs). In this paper we use an NER which allows a partially annotated corpus as a training data as well as a normal fully annotated corpus (Mori et al., 2012).[4] In the training step this NER estimates the parameters of a classifier based on logistic regression (Fan et al., 2008) from sentences fully (or partially) annotated with NEs (concepts). The features are word n-grams surrounding the word in the focus w_i, Table 5 lists the features.

[4]CRFs are also trainable from a partially annotated corpus (Tsuboi et al., 2008). Recently Sasada et al. (2015) have proposed a hybrid method and reported a higher accuracy than CRFs. We may use it for further improvement.

At run-time, given a word sequence, the classifier enumerates all possible BIO2 tags t_i for each word w_i with their probabilities as follows:

$$P_{LR}(t_i|\boldsymbol{w}^-, w_i, \boldsymbol{w}^+),$$

where $\boldsymbol{w}^-$ and $\boldsymbol{w}^+$ are the word sequences preceding it and following it, respectively. Then this NER searches for the tag sequence of the highest probability satisfying the tag sequence constraints.[5]

7 Parsing an Entire Text

The final step is to build a flow graph. The input is a text whose sentences are segmented into words and all the concepts are identified. We call this part a text parsing. As we mentioned in Section 1, text parsing deals with various language phenomena at once, such as dependency, predicate-argument structure, and anaphora/coreference.

For text parsing we extend an MST parser (McDonald et al., 2005). Since the flow graph is a labeled DAG, we add some labeled arcs to the MST. Below we explain the processes one by one.

7.1 Spanning Tree Estimation

We first build a labeled spanning tree covering all the concepts (vertices) of the input text. Let V be a set of vertices and $\mathcal{G}$ be a set of possible spanning trees on V. We assume that there exists a score function $s(u, v, l)$ which represents the likelihood of making a labeled arc from u to v with label l. Then the maximum spanning tree (MST) can be found as follows:

$$\hat{G} = \mathop{\textbf{argmax}}_{G \in \mathcal{G}} \sum_{(u,v,l) \in G} s(u, v, l).$$

We solve this problem using the Chu-Liu-Edmonds algorithm (Chu and Liu, 1965; Edmonds, 1967). We define the score function $s(u, v, l)$ as a probability[6]:

$$s(u, v, l) = \frac{\exp\{\boldsymbol{\Theta} \cdot \boldsymbol{f}(u, v, l)\}}{\displaystyle\sum_{(x,r) \in (V \setminus \{u\}) \times L} \exp\{\boldsymbol{\Theta} \cdot \boldsymbol{f}(u, x, r)\}}.$$

Here L is the arc label set (See Table 3), $\boldsymbol{\Theta}$ is a vector of weight parameters and $\boldsymbol{f}(u, v, l)$ is a function that maps a labeled arc into a feature vector. The score function $s(u, v, l)$ computes the probability of making a labeled arc from u to v with label l referring to their word sequences, concept tags, surrounding words in the original recipe text, and label l. A detailed description is given in Subsection 7.3.

We use a log-linear model (Berger et al., 1996) in order to train the weight parameters $\boldsymbol{\Theta}$. Let $\{(V_t, u_t, v_t, l_t)\}_{t=1}^{T}$ denote a set of T training instances, where V_t is a set of the vertices and (u_t, v_t, l_t) is a gold standard arc with label l in the t-th training instance. Given these training examples, weight parameters are estimated so that they maximize the following likelihood:

$$\sum_{t=1}^{T} \log \frac{\exp(\boldsymbol{\Theta} \cdot \boldsymbol{f}(u_t, v_t, l_t))}{\displaystyle\sum_{(x,r) \in (V_t \setminus \{u_t\}) \times L} \exp(\boldsymbol{\Theta} \cdot \boldsymbol{f}(u_t, x, r))} - \frac{1}{2}\|\boldsymbol{\Theta}\|^2.$$

Because our flow graph is not a tree but a DAG, there can be more than one arcs outgoing from a single vertex. In other words it may contain two arcs, (u, v, l) and (u, v', l'), which share the same start vertex u. In these cases, we add both (V, u, v, l) and (V, u, v', l') to the training data.

7.2 Arc Addition

Given the labeled MST G, we add some labeled arcs to form a flow graph of a labeled DAG with a root. Our algorithm for adding arcs is described in Figure 2.

```
1:  $G \leftarrow$ Maximum spanning tree of $V$.
2:  $A \leftarrow$ Sequence of arcs that can be added to $G$
    without violating the acyclic condition.
3:  Sort $A$ in the descending order of the value of
    the score function $s$.
4:  $n \leftarrow 1$
5:  for $(u, v) \in A$ do
6:    if $G \cup \{(u, v)\}$ is acyclic and $p(n) < s(u, v)$ then
7:      $G \leftarrow G \cup \{(u, v)\}$
8:      $n \leftarrow n + 1$
9:    end if
10: end for
11: return $G$
```

Figure 2: Algorithm of DAG estimation

The most important point is to choose the best arcs one by one among those which do not create cycles and add them to the MST. In Figure 2

[5]For example, the tag sequence F-I T-I is invalid.

[6]This is the probability of a directed arc with label l from a fixed vertex u, but not a probability over all the directed arcs. We have tried the latter scoring function but the result was worse than the former scoring function which we report in this paper.

$s(v, w, l)$ is the same score function used in the MST estimation and $p(n)$ is a function that gives a penalty when the n-th additional arc is added to G. Thus the arc of the highest s is added if the s value is larger than the penalty $p(n)$.

We adopt an exponentially increasing function as the penalty function $p(n)$ with parameters λ and ξ as follows[7]:

$$p(n) = \frac{\xi}{\lambda e^{-\lambda n}}.$$

The denominator on the right-hand side is an exponential distribution with parameter λ. The numerator ξ is a scale parameter. At the training step we first estimate λ which minimizes the square error between the training-data distribution of the number of arcs added to the tree and the exponential distribution. Then we choose ξ which maximizes the F-measure on the held-out data, a small portion of training data, as we do in the deleted interpolation method (Jelinek, 1985).

7.3 Features

The feature function outputs a high dimensional feature vector that represents a characteristic of a labeled arc (u, v, l).

We extract features from labeled arcs (u, v, l) by two processes: first we extract features from the arc and input recipe text and then we concatenate the label l to each feature we extract. First the following features are extracted from the input arc (u, v) and the recipe text:

F1: The number of concepts existing between u and v with sign, which is +1 if u is left to v and -1 otherwise,

F2: Whether u and v are in the same sentence,

F3: Whether u and v are in the same step,

F4: Word sequences, pronunciation and concept tags of each u and v,[8]

F5: Three preceding words and three following words for both u and v,

F6: concept tag of u $\wedge$ concept tag of v,

[7]The reason is that, in small training data, the relationship between the number of additional arcs and the number of the flow graph matched with an exponentially decreasing function well.

[8]The pronunciations are automatically estimated based on the method described in (Mori and Neubig, 2011).

Method	Precision	Recall	F-measure
Baseline	65.1	61.5	63.2
Proposed	73.5	69.1	71.2

Table 6: Accuracies of the baseline and proposed methods.

F7: concept tag of u $\wedge$ concept tag of v $\wedge$ whether u and v are in the same sentence,

F8: concept tag of u $\wedge$ concept tag of v $\wedge$ whether Ac exists between u and v $\wedge$ whether a function word exists between u and v,

F9: concept tag of u $\wedge$ concept tag of v $\wedge$ function words between u and v.

Here the symbol $\wedge$ indicates the combination of individual features.

Next we simply concatenate the label l with each feature to construct feature vectors of labeled arcs. For example, we extracts a feature "concept tag of u $\wedge$ concept tag of v." Then, this type of feature becomes "l $\wedge$ concept tag of u $\wedge$ concept tag of v." by concatenating the label l. The same concatenation of a label is done on the other features.

So-called higher order features which refer to the neighboring vertices in the DAG are common in work on dependency parsing (McDonald et al., 2006; Koo and Collins, 2010), but we do not use these kinds of features because we only have 200 recipes annotated with DAGs[9]. This number is much smaller than roughly 40,000 sentences of the Wall Street Journal which are commonly used to train dependency parsers (Marcus et al., 1994).

8 Evaluation

We evaluated our framework on the r-FG corpus described in Table 1. We executed 10-fold cross validation for more reliable results.

DAG estimation accuracy is measured by the F-measure of labeled arcs, which is the harmonic mean of precision and recall. Let N_{sys}, N_{ref}, and N_{int} be the number of the estimated arcs, the gold standard arcs, and their intersection, respectively. Then precision = N_{int}/N_{sys}, recall = N_{int}/N_{ref}, and F-measure = $2N_{int}/(N_{ref} + N_{sys})$.

[9]Mori et al. (2014) state that it took about one our to annotate one recipe with word boundaries, concept tags, and a flow graph. It is much more coslty than syntactic annotation.

Task	Input	F-meas.
WS	Raw texts	98.6
CI	Gold WS results	90.7
Flow graph estimaiton	Gold WS/CI results	72.1
WS + CI + flow graph estimaiton	Raw texts	51.6

Table 7: F-measure of each task and the overall task.

8.1 Flow Graph Estimation

As the first evaluation, we compared the simple application of the MST parser and our extension. We assumed the gold WS and CI results.

8.1.1 Settings of Flow Graph Estimation

We compared two methods in the following way.

Baseline A naive method for the text parsing is a simple application of MST parser (McDonald et al., 2005) to a concept sequence input. An MST parser takes a sequence of words annotated with POSs and outputs a labeled tree connecting all the words. Thus our baseline for flow graph estimation takes a sequence of concepts (pairs of a word sequence and a concept type) as the input. The output of an MST parser is a tree, but not a DAG. So we add our arc addition module for a fair comparison. As the implementation, we modified a Japanese dependency parser (Flannery et al., 2012) that uses the logistic regression as the scoring function.

Proposed This combines the spanning tree estimation (Subsection 7.1) and the arc addition (Subsection 7.2) in the cascaded manner. Different from the **Baseline**, this method referred to words not included in concepts such as function words as the features .

Table 6 shows the accuracies of the baseline method and our MST extension. We can see that there is a significant difference in accuracy between **Baseline** and **Proposed**. The major difference between these two methods is whether or not they refer to the words not covered by the concepts in the original texts, such as in **F5** for example. These words are mainly function words. Therefore we can say that the function words are, as we know intuitively, important for the relationships among the concepts.

8.2 Text Parsing on a Raw Text

We also executed the text parsing taking a raw text as the input. For this problem, we combine WS, CI, and flow graph estimation (**Proposed**) in the cascaded manner.

The performance measurement is F-measure. Different from the first experiment, the unit is a triplet $(\langle \boldsymbol{w}_s, c_s\rangle, \langle \boldsymbol{w}_e, c_e\rangle, l)$. Here, $\boldsymbol{w}_s$ and c_s are the word sequence of the out-going vertex of the arc and its concept type, respectively. $\boldsymbol{w}_e$ and c_e are those of its in-coming vertex. And l is its label. A triplet is correct if and only if all these elements match with those of an arc in the manually annotated data.

8.2.1 Settings of Word Segmentation and Concept Identification

We built an automatic word segmenter and an automatic concept identifier in the following way.

WS: The word segmenter (see Section 5) is trained on the following corpora.

1. Balanced Corpus of Contemporary Written Japanese (Maekawa, 2008) containing fully segmented 53,899 sentences from newspaper articles, books, magazines, whitepapers, Web logs, and Web QAs.
2. The partially segmented sentences derived from 208 recipes in the r-FG corpus and additional 208 recipes annotated with concept types. In the experiment, we excluded the test part in 10-fold cross validation. Thus we built 10 models in total.
3. Partially annotated 1,651 sentences crawled from another recipe Web site[10].

CI: The concept identifier (see Section 6) is trained on the corpus 2. used in the WS training in the same way. Thus we built 10 models in total for 10-fold cross validation[11].

[10]`http://park.ajinomoto.co.jp/` accessed on 2014/May/21.

[11]We made this concept identifier with the model trained on 416 recipes publicly available from `http://plata.ar.media.kyoto-u.ac.jp/data/recipe/home-e.html`.

8.2.2 Results

Table 7 shows the accuracies of WS, CI, and flow graph estimation on the gold results of the preceding task and that of the combination of three tasks starting from a raw text to a flow graph.

As we see in the table, the flow graph estimation task is the most difficult and has a large room for improvement. The accuracy of WS without adaptation was about 95% and our adaptation technique raised it to about 99% which is as high as in the general domain case. The accuracy of CI, trained on less than 3,000 sentences, is as high as the general NER whose accuracy is about 90% by a model trained on about 10,000 sentences. This is still lower than WS accuracy, so the concept identifier is also a target of improvement.

From Table 7, the accuracy of the cascade combination of three tasks (WS + CI + flow graph estimation) is 51.6. This value is lower than the simple multiplication result that assumes the independence among the tasks $57.2 = (0.986^{1.27} \times 0.907)^2 \times 0.721 \times 100$, where 1.27 is the average word length of the concepts. This indicates that it is worth trying to investigate joint methods for WS, CI, and flow graph estimation.

9 Conclusion

In this paper, we proposed a framework of procedural text understanding consisting of the three processes. The first process is the well-known word identification. The second one is concept identification, which can be solved in the same way of named entity recognition with different definition of named entities. The third one is our original text parsing, which estimates a flow graph taking a text and the concepts in it as the input.

We tested our framework on recipe texts manually annotated with a flow graph. The results showed that our method outperforms a naive application of an MST dependency parser. Thus we can say that the simple application of dependency parsing to flow graph estimation does not work well, and that it is important to focus on not only concepts but also words surrounding them. Finally we evaluated the full automatic process of building a flow graph from a raw text. Our result can be a solid baseline for future improvement in the procedural text understanding problem.

Our method is applicable to various procedural texts allowing us to realize more intelligent search engine for how-to texts, more sophisticated symbol grounding by combining NLP and CV, etc.

Acknowledgments

This work was supported by JSPS Grants-in-Aid for Scientific Research Grant Numbers 26280084, 26280039, and 24240030.

References

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

Andrew Borthwick. 1999. *A Maximum Entropy Approach to Named Entity Recognition*. Ph.D. thesis, New York University.

Sabine Buchholz and Erwin Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, pages 149–164.

Nancy A. Chinchor. 1998. Overview of MUC-7/MET-2. In *Proceedings of the Seventh Message Understanding Conference*.

Yoeng-Jin Chu and Tseng-Hong Liu. 1965. On the shortest arborescence of a directed graph. *Science Sinica*, 14:1396–1400.

Jack Edmonds. 1967. Optimum branchings. *Journal Research of the National Bureau of Standards*, 71B:233–240.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. 2008. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874.

Raquel Fernández, Jonathan Ginzburg, and Shalom Lappin. 2004. Classifying ellipsis in dialogue: A machine learning approach. In *Proceedings of the 20th International Conference on Computational Linguistics*, pages 240–246.

Daniel Flannery, Yusuke Miyao, Graham Neubig, and Shinsuke Mori. 2012. A pointwise approach to training dependency parsers from partially annotated corpora. *Journal of Natural Language Processing*, 19(3).

Reiko Hamada, Ichiro Ide, Shuichi Sakai, and Hidehiko Tanaka. 2000. Structural analysis of cooking preparation steps in Japanese. In *Proceedings of the fifth International Workshop on Information Retrieval with Asian Languages*, pages 157–164.

Atsushi Hashimoto, Naoyuki Mori, Takuya Funatomi, Yoko Yamakata, Koh Kakusho, and Michihiko Minoh. 2008. Smart kitchen: A user centric cooking support system. In *Proceedings of the 12th Information Processing and Management of Uncertainty in Knowledge-Based Systems*, pages 848–854.

Atsushi Hashimoto, Tetsuro Sasada, Yoko Yamakata, Shinsuke Mori, and Michihiko Minoh. 2014. Kusk dataset: Toward a direct understanding of recipe text and human cooking activity. In *Proceedings of the SixthInternational Workshop on Cooking and Eating Activities*.

Frederick Jelinek. 1985. Self-organized language modeling for speech recognition. Technical report, IBM T. J. Watson Research Center.

Terry Koo and Michael Collins. 2010. Efficient third-order dependency parsers. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1–11.

Kikuo Maekawa. 2008. Balanced corpus of contemporary written Japanese. In *Proceedings of the 6th Workshop on Asian Language Resources*, pages 101–102.

Mitchell Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. 1994. The penn treebank: Annotating predicate argument structure. In *Proceedings of the ARPA Workshop on Human Language Technology*, pages 114–119.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 523–530.

Ryan McDonald, Kevin Lerman, and Fernando Pereira. 2006. Multilingual dependency analysis with a two-stage discriminative parser. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, pages 216–220.

Bernard Merialdo. 1994. Tagging English text with a probabilistic model. *Computational Linguistics*, 20(2):155–171.

Yoshio Momouchi. 1980. Control structures for actions in procedural texts and PT-chart. In *Proceedings of the Eighth International Conference on Computational Linguistics*, pages 108–114.

Shinsuke Mori and Graham Neubig. 2011. A pointwise approach to pronunciation estimation for a TTS front-end. In *Proceedings of the InterSpeech2011*, pages 2181–2184, Florence, Italy.

Shinsuke Mori and Graham Neubig. 2014. Language resource addition: Dictionary or corpus? In *Proceedings of the Nineth International Conference on Language Resources and Evaluation*, pages 1631–1636.

Shinsuke Mori, Tetsuro Sasada, Yoko Yamakata, and Koichiro Yoshino. 2012. A machine learning approach to recipe text processing. In *Proceedings of the 1st Cooking with Computer Workshop*, pages 29–34.

Shinsuke Mori, Hirokuni Maeta, Yoko Yamakata, and Tetsuro Sasada. 2014. Flow graph corpus from recipe texts. In *Proceedings of the Nineth International Conference on Language Resources and Evaluation*, pages 2370–2377.

Iftekhar Naim, Young Chol Song, Qiguang Liu, Henry Kautz, Jiebo Luo, and Daniel Gildea. 2014. Unsupervised alignment of natural language instructions with video segments. In *Proceedings of the 28th National Conference on Artificial Intelligence*.

Graham Neubig, Yosuke Nakata, and Shinsuke Mori. 2011. Pointwise prediction for robust, adaptable Japanese morphological analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, pages 529–533.

Leif Arda Nielsen. 2004. Verb phrase ellipsis detection using automatically parsed text. In *Proceedings of the 20th International Conference on Computational Linguistics*, pages 1093–1099.

Vignesh Ramanathan, Percy Liangy, and Li Fei-Fei. 2013. Video event understanding using natural language descriptions. In *Proceedings of the 14th International Conference on Computer Vision*.

Michaela Regneri, Marcus Rohrbach, Dominikus Wetzel, Stefan Thater, Bernt Schiele, and Manfred Pinkal. 2013. Grounding action descriptions in videos. *Transactions of the Association for Computational Linguistics*, 1:25–36.

Marcus Rohrbach, Wei Qiu, Ivan Titov, Stefan Thater, Manfred Pinkal, and Bernt Schiele. 2013. Translating video content to natural language descriptions. In *Proceedings of the 14th International Conference on Computer Vision*.

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Computational Natural Language Learning*, pages 142–147.

Tetsuro Sasada, Shinsuke Mori, Tatsuya Kawahara, and Yoko Yamakata. 2015. Named entity recognizer trainable from partially annotated data. In *Proceedings of the Eleventh International Conference Pacific Association for Computational Linguistics*.

Hirotoshi Taira, Sanae Fujita, and Masaaki Nagata. 2010. Predicate argument structure analysis using transformation-based learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*.

Yuta Tsuboi, Hisashi Kashima, Shinsuke Mori, Hiroki Oda, and Yuji Matsumoto. 2008. Training conditional random fields using incomplete annotations. In *Proceedings of the 22nd International Conference on Computational Linguistics*.

Liping Wang, Qing Li, Na Li, Guozhu Dong, and Yu Yang. 2008. Substructure similarity measurement in Chinese recipes. In *Proceedings of the 17th International Conference on World Wide Web*, pages 978–988.

Yuk Wah Wong and Raymond Mooney. 2007. Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics*, pages 960–967.

Yoko Yamakata, Shinji Imahori, Yuichi Sugiyama, Shinsuke Mori, and Katsumi Tanaka. 2013. Feature extraction and summarization of recipes using flow graph. In *Proceedings of the 5th International Conference on Social Informatics*, LNCS 8238, pages 241–254.

Koichiro Yoshino, Shinsuke Mori, and Tatsuya Kawahara. 2013. Predicate argument structure analysis using partially annotated corpora. In *Proceedings of the Sixth International Joint Conference on Natural Language Processing*.

Luke S. Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of Uncertainty in Artificial Intelligence*.

Suitability of ParTes Test Suite for Parsing Evaluation

Marina Lloberes
U. de Barcelona
Barcelona, Spain
mllobesa8@alumnes.ub.edu

Irene Castellón
U. de Barcelona
Barcelona, Spain
icastellon@ub.edu

Lluís Padró
U. Politècnica de Catalunya
Barcelona, Spain
padro@cs.upc.edu

Abstract

Parsers have evolved significantly in the last decades, but currently big and accurate improvements are needed to enhance their performance. ParTes, a test suite in Spanish and Catalan for parsing evaluation, aims to contribute to this situation by pointing to the main factors that can decisively improve the parser performance.

1 Introduction

Parsing has been a very active area, so that parsers have progressed significantly over the recent years (Klein and Manning, 2003; Collins and Koo, 2005; Nivre et al., 2006; Ballesteros and Nivre, 2012; Bohnet and Nivre, 2012; Ballesteros and Carreras, 2015). However, nowadays significant improvement in parser performance needs extra effort.

A deeper and detailed analysis of the parsers performance can provide the keys to exceed the current accuracy. Tests suites are a linguistic resource which makes it possible this kind of analysis and which can contribute to highlight the key issues to improve decisively the Natural Language Processing (NLP) tools (Flickinger et al., 1987; EAGLES, 1994; Lehmann et al., 1996).

This paper presents ParTes 15.02, a test suite of syntactic phenomena for parsing evaluation. This resource contains an exhaustive and representative set of structure and word order phenomena for Spanish and Catalan languages (Lloberes et al., 2014). The new version adds a development data set and a test data set.

The rest of the paper describes the main contributions in test suite development (Section 2). Section 3 shows the characteristics and the specifications of ParTes. The results of an evaluation task of the FreeLing Dependency Grammars (FDGs) with verb subcategorization information added (Lloberes et al., 2010) using ParTes are discussed in Section 4. Finally, the main conclusions and future work are exposed (Section 5).

Features	**HP**	**EAGLES**	**TSNLP**
Domain	general	specific	general
Goal	parsing	grammar checkers	NLP software
Languages	English	English	English, German, French
Annotation	minimal	minimal	robust
Content	syntax	taxonomy of errors	(extra-)linguistic

Table 1: HP, EAGLES & TSNLP features

2 Test suite development

The main aim of qualitative studies is to offer empirical evidence about the richness and precision of the data, in comparison with quantitative studies which provide a view of the actual spectrum (McEnery and Wilson, 1996). For this reason, qualitative analysis are deep and detail-oriented, while quantitative analysis focus on statistically informative data. In the qualitative approach, representativeness of the studied phenomena focuses on exhaustiveness rather than frequency, which is the base of the quantitative approach. Both approaches are not exclusive because they contribute to build a global interpretation.

While corpora are a large databases of the most frequent linguistic utterances (McEnery and Wilson, 1996), test suites are controlled and exhaustive databases of linguistic utterances classified by linguistic features. These collections of cases are internally organized and richly annotated (Lehmann et al., 1996). Controlledness, exhaustiveness and detailedness properties allow these databases to provide qualitatively analyzed data.

They were developed in parallel with the NLP technologies. The more sophisticated the software became, the more complex the test suites evolved to be (Lehmann et al., 1996). From a collection of interesting examples, they transformed into deeply structured and richly annotated databases (Table 1), such as the HP test suite (Flickinger et al., 1987), the test suite developed by one of the groups of EAGLES (EAGLES, 1994), the TSNLP (Lehmann et al., 1996) and the corpus of unbounded depdendencies (Rimell et al., 2009).

Concerning the languages of this study, a test suite for Spanish was developed by Marimon et al. (2007). The goal of this test suite is to assess the development of a Spanish Head-Driven Phrase Structure Grammar and it offers grammatical and agrammatical test cases.

```
<level name="intrachunk">
   <constituent name="nounphrase">
      <hierarchy name="child">
         <realization id="0037"
            name="prepositionalphrase"
            class="noun" subclass="prepobj"
            link="n-s" freq="0.084357"
            parent_devel="recurso"
            child_devel="para"
            parent_test="libro"
            child_test="para"
            devel="Es un recurso para los
            alumnos"
            test="Los alumnos tienen un
            libro para la lectura"/>
      </hierarchy>
   </constituent>
</level>
```

Figure 1: Structure in ParTes. Example of the PP-attachment in the noun phrase

3 The ParTes test suite

This test suite is a hierarchically structured and richly annotated set of of syntactic phenomena for qualitative parsing evaluation available in Spanish (ParTesEs) and Catalan (ParTesCa) and freely distributed under the Creative Commons Attribution-ShareAlike 3.0 Unported License.[1]

The new release of ParTes (15.02) consists in the improvement of the linguistic data sets. Initially, ParTes included a test data module formed by sentences illustrating the syntactic phenomena of the test suite (Lloberes et al., 2014). The current version incorporates a set of linguistic data for development purposes that extends the capabilities of the test suite by allowing the parser development monitoring and a second iteration of the evaluation task.

This resource has been created following the main contributions in test suite design (Flickinger et al., 1987; EAGLES, 1994; Lehmann et al., 1996). The main feature shared with the existent test suites is the control over the data, which makes it possible to work as a qualitative evaluation tool. Furthermore, ParTes adds the concepts of complexity of the resource organization, exhaustiveness of the phenomena descriptions and representativity of the phenomena included.

ParTes is a test suite of syntactic phenomena annotated with syntactic and meta-linguistic information. The content has been hierarchically structured by means of syntactic features and over two major syntactic concepts (Figures 1 and 2): structure and word order.

It provides an exhaustive description of the syntactic phenomena, offering a detailed view of their features and their behavior. A selection of the representative phenomena has been performed, which allowed to delimit the number of cases preserving the control over the data.

The test suite has been semi-automatically generated, extracting automatically data from computational resources when available. Otherwise, written linguistic resources have been used to populate manually the resource. Its architecture makes it possible to extend the test suite to new languages, although the current version is available in two languages.

3.1 Test suite specifications

The current version contains a total of 161 syntactic phenomena in ParTesEs (99 relate to syntactic structure and 62 to word order) and a total of 145 syntactic phenomena in ParTesCa (99 concern to syntactic structure and 46 to word order).

The structure phenomena have been manually collected from descriptive grammars (Bosque and Demonte, 1999; Solà et al., 2002) and represented following the criteria of the FDGs (Lloberes et al., 2010). The selection of phenomena has been validated by the dependency links frequency of the AnCora Corpus (Taulé et al., 2008).

As Figure 1 shows, the first level of the hierarchy determines the *level* of the syntactic phenomenon (inside a chunk or between a marker and the subordinate verb). The second level expresses the phrase or the clause involved in the syntactic phenomenon (*constituent*) and the third level describes the position (*head* or *child*) in the *hierarchy*. Finally, a set of syntactic features describes the type of constituent observed (*realization*).

Specifically, the syntactic features of the *realization* concern to the grammatical category, the

[1]http://grial.uab.es/recursos.php

```
<class name="subj#V">
   <schema name="subj#V">
      <realization id="0104"
         func="subj#v"
         cat="pron#v"
         parent="perdre"
         children="tot"
         constr="passive-pron"
         sbjtype="full"
         freq="0.001875"
         idsensem="45074#45239#48770"
         test="Tot s'ha perdut"/>
   </schema>
</class>
```

Figure 2: Word order in ParTes. Example of pronominal passive with particle 'se'

phrase or the clause that defines the structure phenomenon (*name*), its syntactic specifications (*class*, *subclass*), the arch between the parent and the child (*link*), the occurrence frequency of the link (*freq*) in the AnCora Corpus. Additionally, every phenomenon is identified with a numeric *id*.

For every syntactic structure phenomenon, two linguistic examples have been manually defined, one of them to be used for development purposes (*devel*) and the other one for testing purposes (*test*). The lemmas of the parent and the child of the exemplified phenomenon are also provided (*parent_devel*, *parent_test*, *child_devel*, *child_test*).

Word order in ParTes is semi-automatically built from the most frequent argument structure frames of the SenSem Corpus (Fernández and Vàzquez, 2014).

The hierarchy about the word order is structured firstly by the number and the type of arguments of the word order schema (*class*), as Figure 2 illustrates. Every class is defined by a set of *schemas* about the number of arguments and their order. The most concrete level (*realization*) describes the properties of the schema.

These properties refer to the syntactic function (*func*)[2] and the grammatical category (*cat*) of every argument of the schema. Furthermore, the type of construction (*constr*) where the schema occurs in and the type of subject (*sbjtype*) are provided. The occurrence frequency of the schema in the SenSem Corpus is associated (*freq*). In addition, a numeric *id* is assigned to every schema and a link to SenSem Corpus sentences with the same schema is created (*idsensem*).

Every schema recorded is exemplified with a sentence for testing purposes (*test*). For every test sentence, the lemmas of the *parent* and the *children* corresponding to the head of the arguments of the schema are added.

3.2 Description of the data sets

The development and the test data are built over the manually defined linguistic examples of the syntactic phenomena of ParTes.

The sentences have been automatically annotated by using the FDGs, so that a complete dependency analysis of the whole sentence is offered. The output has been reviewed manually by two annotators: a native in Spanish responsible for the annotation of ParTesEs and a native in Catalan who annotated the ParTesCa. A second manual revision has been performed: the Catalan annotator reviewed the ParTesEs annotated and the Spanish annotator reviewed the ParTesCa annotated guaranteeing the agreement between the annotations in both languages and preserving the quality of the annotation according to the criteria.

Up to the current version, the number of sentences referring to the syntactic structure are: 95 sentences in the ParTesEs development data set, 99 sentences in the ParTesEs test data set, 98 sentences in the ParTesCa development data set and 99 sentences in the ParTesCa test data set. The data sets are distributed in plain text format and in the CoNLL annotation format (Nivre et al., 2007).

4 Evaluation task

In order to test the usability of ParTes for parsing evaluation, it has been applied as a gold standard in an evaluation task of the FDGs. Particularly, the capabilities of the test suite have been tested for explaining the performance of FDG as regards the argument recognition since it still remains to be solved successfully (Carroll et al., 1998; Zeman, 2002; Mirroshandel et al., 2013).

The FDGs are the core part of the rule-based FreeLing Dependency Parser (Padró and Stanilovsky, 2012). They provide a deep and complete syntactic analysis in the form of dependencies. The grammars are a set of manually-defined rules that comple the structure of the tree (*linking rules*) and assign a syntactic function to every link of the tree (*labelling rules*) by means of a system of priorities and a set of conditions.

Two FDGs versions for both languages have been evaluated: a version without verb subcategorization classes (*Bare*) and a version with verb sub-

[2]Tagset: *adjt* - adjunct; *attr* - attribute; *dobj* - direct object; *iobj* - indirect object; *pobj* - prepositional object; *pred* - predicative; *subj* - subject.

	ParTesEs		ParTesCa	
Metric	**Bare**	**Subcat**	**Bare**	**Subcat**
LAS	77.57	79.66	79.41	81.80
UAS	88.21	88.21	88.24	88.24
LA	78.90	81.94	80.88	83.64

Table 2: Label Accuracy of FDG on ParTes

categorization classes (*Subcat*) extracted from the verbal frames of the SenSem Corpus (Fernández and Vàzquez, 2014). The system analysis built for every version of the grammars is compared to the ParTes analysis using the evaluation metrics of the CoNLL-X Shared Task (Nivre et al., 2007).[3]

According to the accuracy results (Table 2), the evaluation with ParTes shows that FDGs performance is medium-accuracy (near or above 80% in LAS). Both versions of the grammar in both languages perform in high-accuracy in terms of attachment (UAS), whereas they obtain medium accuracy on syntactic function labelling (LA). ParTes data highlight that the *Subcat* grammar scores better than the *Bare* grammar in LA, which is directly related to the addition of subcategorization classes, as stated in the following discussion.

A detailed observation reveals that ParTes sentences related to subcategorization are performed better in precision by *Subcat* rather than *Bare* (Table 3). Furthermore, the test data allows to show that subcategorization has more impact in the recognition of the majority of arguments (*dobj*, *pobj*, *pred*) and the subject (*subj*) than in the adjuncts (*adjt*) because the precision scores increment is higher. Subcategorization do not have an effect on the attribute (*attr*) because it can be solved lexically. The indirect objects (*iobj*) correspond to cases of dative clitic, which are solved by morphological information.

The integration of subcategorization information bounds the rules to the verbs included in the classes. Consequently, some cases may be not captured if the verb is not expected by the subcategorization classes as it happens in the prepositional object (*pobj*). For example, the prepositional argument of the sentence 'Ha creido en sí mismo' ('He has believed in himself') should be labelled as *pobj*, but the *adjt* tag is assigned because the verb 'creer' is not in any of the prepositional argument classes of the grammar. However, in the majority of types of arguments and the adjuncts the recall is maintained or increased (Table 4).

[3] *Labeled Attachment Score* (LAS): the percentage of tokens with correct head and syntactic function label; *Unlabeled Attachment Score* (UAS): the percentage of tokens with correct head; *Label Accuracy* (LA): the percentage of tokens with correct syntactic function label; *Precision* (P): the ratio between the system correct tokens and the system tokens; *Recall* (R): the ratio between the system correct tokens and the gold standard tokens.

	ParTesEs			ParTesCa		
Tag	**#**	**Bare**	**Subcat**	**#**	**Bare**	**Subcat**
adjt	39	53.85	65.96	30	60.00	61.90
attr	28	88.89	83.87	20	90.00	78.26
dobj	39	65.31	73.81	42	74.51	86.96
iobj	7	100.00	100.00	3	100.00	75.00
pobj	11	23.68	37.50	13	45.83	60.00
pred	2	25.00	100.00	2	22.22	100.00
subj	51	93.02	93.02	43	87.88	90.62

Table 3: Precision scores of FDG on ParTes

	ParTesEs			ParTesCa		
Tag	**#**	**Bare**	**Subcat**	**#**	**Bare**	**Subcat**
adjt	39	35.90	79.49	30	50.00	86.67
attr	28	85.71	92.86	20	90.00	90.00
dobj	39	82.05	79.49	42	90.48	95.24
iobj	7	28.57	28.57	3	66.67	100.00
pobj	11	81.82	54.55	13	84.62	69.23
pred	2	50.00	50.00	2	100.00	50.00
subj	51	78.43	78.43	43	67.44	67.44

Table 4: Recall scores of FDG on ParTes

5 Conclusions

The new version of the ParTes test suite for parsing evaluation has been presented. The main features and the data sets have been described. In addition, the results of an evaluation task of the FDGs with ParTes data have been exposed.

The characteristics of the test suite made it possible to analyze in detail the causes of the performance improvement on the argument recognition of the FDGs including subcategorization information. Therefore, these results show that ParTes is an appropriate resource for parsing evaluation.

Currently, ParTes is extended to English following the methodology explained in this paper. In the upcoming releases, test and development sentences belonging to the word order will be incorporated in the ParTes data sets. Furthermore, we are exploring a systematic methodology to generate agrammatical variants of the existent sentences.

Acknowledgments

This work has been funded by the SKATeR project (Spanish Ministry of Economy and Competitiveness, TIN2012-38584-C06-06 and TIN2012-38584-C06-01).

References

M. Ballesteros and X. Carreras. 2015. Transition-based Spinal Parsing. In *Proceedings of CoNLL-2015*.

M. Ballesteros and J. Nivre. 2012. MaltOptimizer: A System for MaltParser Optimization. In *Proceedings of the Eight International Conference on Language Resources and Evaluation*.

B. Bohnet and J. Nivre. 2012. A Transition-based System for Joint Part-of-speech Tagging and Labeled Non-projective Dependency Parsing. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*.

I. Bosque and V. Demonte. 1999. *Gramática Descriptiva de la Lengua Española*. Espasa Calpe.

J. Carroll, G. Minnen, and T. Briscoe. 1998. Can Subcategorisation Probabilities Help a Statistical Parser? In *Proceedings of the 6th ACL/SIGDAT Workshop on Very Large Corpora*.

M. Collins and T. Koo. 2005. Discriminative Reranking for Natural Language Parsing. *Computational Linguistics*, 31(1).

EAGLES. 1994. Draft Interim Report EAGLES. Technical report, Expert Advisory Group on Language Engineering Standards.

A. Fernández and G. Vàzquez. 2014. The SenSem Corpus: an annotated corpus for Spanish and Catalan with information about aspectuality, modality, polarity and factuality. *Corpus Linguistics and Linguistic Theory*, 10(2).

D. Flickinger, J. Nerbonne, and I.A. Sag. 1987. Toward Evaluation of NLP Systems. Technical report, Hewlett Packard Laboratories. Distributed at the 24th Annual Meeting of the Association for Computational Linguistics (ACL).

D. Klein and C.D. Manning. 2003. Accurate Unlexicalized Parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1*.

S. Lehmann, S. Oepen, S. Regnier-Prost, K. Netter, V. Lux, J. Klein, K. Falkedal, F. Fouvy, D. Estival, E. Dauphin, H. Compagnion, J. Baur, L. Balkan, and D. Arnold. 1996. TSNLP - Test Suites for Natural Language Processing. In *Proceedings of the 16th Conference on Computational Linguistics*, volume 2.

M. Lloberes, I. Castellón, and L. Padró. 2010. Spanish FreeLing Dependency Grammar. In *Proceedings of the Seventh Conference on International Language Resources and Evaluation*.

M. Lloberes, I. Castellón, L. Padró, and E. Gonzàlez. 2014. ParTes. Test Suite for Parsing Evaluation. *Procesamiento del Lenguaje Natural*, 53.

M. Marimon, N. Bel, and N. Seghezzi. 2007. Test-suite Construction for a Spanish Grammar. In *Proceedings of the GEAF 2007 Workshop*.

T. McEnery and A. Wilson. 1996. *Corpus Linguistics*. Edinburgh University Press, Edinburgh.

S.A. Mirroshandel, A. Nasr, and B. Sagot. 2013. Enforcing Subcategorization Constraints in a Parser Using Sub-parses Recombining. In *NAACL 2013 - Conference of the North American Chapter of the Association for Computational Linguistics*.

J. Nivre, J. Hall, J. Nilsson, G. Eryiğit, and S. Marinov. 2006. Labeled Pseudo-projective Dependency Parsing with Support Vector Machines. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*.

J. Nivre, J. Hall, S. Kübler, R. McDonald, J. Nilsson, S. Riedel, and D. Yuret. 2007. The CoNLL 2007 Shared Task on Dependency Parsing. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*.

L. Padró and E. Stanilovsky. 2012. Freeling 3.0: Towards wider multilinguality. In *Proceedings of the Eight International Conference on Language Resources and Evaluation*.

L. Rimell, S. Clark, and M. Steedman. 2009. Unbounded Dependency Recovery for Parser Evaluation. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*.

J. Solà, M.R. Lloret, J. Mascaró, and M. Pérez-Saldanya. 2002. *Gramàtica del Català Contemporani*. Empúries.

M. Taulé, M.A. Martí, and M. Recasens. 2008. AnCora: Multi level annotated corpora for Catalan and Spanish. In *6th International Conference on Language Resources and Evaluation, Marrakesh*.

D. Zeman. 2002. Can Subcategorization Help a Statistical Dependency Parser? In *19th International Conference on Computational Linguistics*.

Coordination-aware Dependency Parsing (Preliminary Report)

Akifumi Yoshimoto[*] **Kazuo Hara**[†] **Masashi Shimbo**[*] **Yuji Matsumoto**[*]

[*]Nara Institute of Science and Technology
Ikoma, Nara, Japan
{akifumi-y,shimbo,matsu}@is.naist.jp

[†]National Institute of Genetics
Mishima, Shizuoka, Japan
kazuo.hara@gmail.com

Abstract

Coordinate structures pose difficulties in dependency parsers. In this paper, we propose a set of parsing rules specifically designed to handle coordination, which are intended to be used in combination with Eisner and Satta's dependency rules. The new rules are compatible with existing similarity-based approaches to coordination structure analysis, and thus the syntactic and semantic similarity of conjuncts can be incorporated to the parse scoring function. Although we are yet to implement such a scoring function, we analyzed the time complexity of the proposed rules as well as their coverage of the Penn Treebank converted to the Stanford basic dependencies.

1 Introduction

Even for state-of-the-art dependency parsers, the recovery of dependencies involving coordinate structures remains difficult. According to Nivre et al. (2010), the accuracy in parsing the construction known as right node raising, which includes a coordinate structure such as "president and chief executive of the company," is less than fifty percent.

Apart from dependency parsers, there are methods specialized for coordination structure analysis, which attempt to identify coordinate structures from the similarity of conjuncts (Kurohashi and Nagao, 1994; Hara and Shimbo, 2007; Hara et al., 2009).

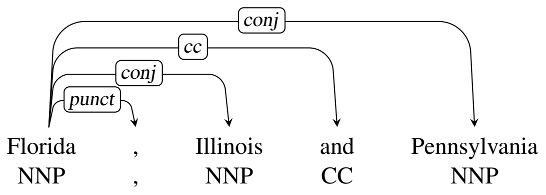
Our final goal is to improve the accuracy of parsing sentences containing coordination, by taking advantage of the above methodologies. In the same vein, Hanamoto et al. (2012) used dual decomposition to combine an HPSG parser with Hara et al.'s (2009) model.

In this paper, we take a different approach: we augment Eisner and Satta's (1999) parsing rules for normal dependencies, with a new set of rules to specifically handle coordinations. This design reflects the fact that Eisner and Satta's "split-head" (Johnson, 2007) rules construct dependency trees on the left and the right of a word independently, and thus the entire span of conjuncts cannot be captured during the course of construction.

Using the Penn Treebank (PTB) (Marcus et al., 1993) after converting the structures to Stanford basic dependency representation (de Marneffe and Manning, 2008), we verified that the proposed rules cover 94% of the sentences involving coordinations. Moreover, about half the failure of the remaining sentences was caused by unusually annotated structures, rather than a weakness in the proposed rules.

2 Dependency structure for coordination

In the Stanford basic dependencies, coordination is represented as a dependency structure in which the first conjunct is the head of the dependency[1], with the second and subsequent conjuncts being the dependents of the first. Commas and coordinate conjunctions (such as "and", "or") separating conjuncts are also the dependents of the first conjunct, with labels *punct* and *cc*, respectively.



Although not explicitly stated in the Stanford manual, in the converted PTB corpus, the first conjunct

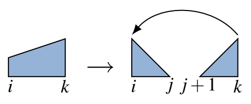
[1]The Stanford dependencies manual (de Marneffe and Manning, 2008) merely states that the first conjunct is *normally* the head, but we take this rule as definitive in this paper. The error analysis of our method based on this assumption is given in Section 5.

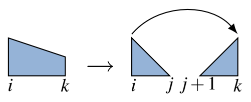
is, in most cases, also the head of extra punctuations (e.g., a comma in front of "and" in "*A*, *B*, and *C*"), or parentheses (e.g., "he says" in "*A*, *B* and, he says, *C*") that occur inside coordinate structure.

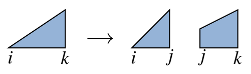
3 The Eisner-Satta algorithm

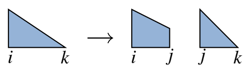
In this section, we briefly review the grammar rules used in the Eisner-Satta algorithm (1999), which our proposed rules are designed to augment. For details about the algorithm, see the original paper, as well as (Koo and Collins, 2010; Johnson, 2007).

The Eisner-Satta algorithm builds a parse tree in a bottom-up manner using the following four rules, each of which yields a span of different type shown next to the respective rule.

(1) INCOMPLETELEFT 

(2) INCOMPLETERIGHT 

(3) CCOMPLETELEFT 

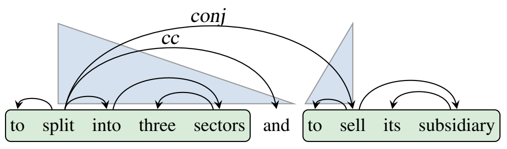
(4) COMPLETERIGHT 

In these rules, *i*, *j*, and *k* indicate word indices. Symbol ℓ is also used for an index later. Let $w(i)$ denote the *i*th word in a sentence. Each word $w(i)$ is initially associated with COMPLETELEFT and COMPLETERIGHT. A root node is added as the leftmost node of a sentence, with only COMPLETERIGHT.

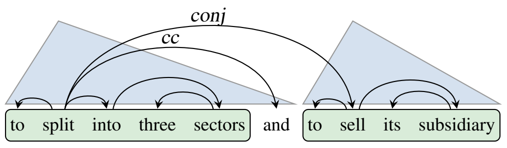
4 Parsing rules for coordination

We augment Eisner and Satta's split-head dependency parsing rules with a set of additional rules that are tailored for similarity evaluation of conjuncts. Split-head rules lead to an efficient $O(n^3)$ parsing algorithm for a sentence of length *n*, but the way in which a parse tree is built does not allow evaluation of conjunct similarity as done in Hara et al. (2009).

When a structure "*A* and *B*" is parsed with the Eisner-Satta algorithm, rule (2) is used for creating a dependency arc (with label *conj*) between conjuncts *A* and *B*. At that moment, the available spans are COMPLETERIGHT whose head is *A*, and COMPLETELEFT whose head is *B*. However, these spans do not correspond to the entire spans of the respective conjuncts. See the figure below for illustration.



Therefore, we introduce new constituents specifically for coordination, which group both sides of the head of conjuncts, as in:

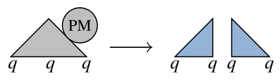


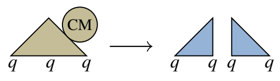
4.1 Punctuations and conjunctions

Let *q*, *r*, and *s* signify word indices whose values are restricted by the position of punctuations or conjunctions[2].

We first group COMPLETELEFT and COMPLETERIGHT for a punctuation or a conjunction as a single constituent. We call the resulting constituents PUNCTMARK (or PM for short) and CCMARK (CM), respectively.

Rule (5) can be applied only if $w(q)$, the word at index *q*, is a commas or a semicolons, and rule (6) only if $w(q)$ is a coordinate conjunctions, such as "and," "or," "and/or," "&," "but," "plus," and "yet."

(5)  $O(p)$

(6)  $O(p)$

As shown next to the rules, the time needed to apply them is $O(p)$, where p ($\ll n$) is the number of coordinate conjunctions and punctuations in a sentence; i.e., the values *q* can potentially take on.

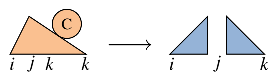
4.2 Rules for cascading conjuncts

In Hara et al. (2009), the plausibility score of a coordinate structure is defined in terms of the similarity between the head conjunct and each of the remaining conjuncts. In the Stanford basic dependencies, the head of a coordinate structure is the first conjunct by default. Thus, to enable computation similar to Hara et al., the end position of the first conjunct must be maintained throughout the

[2]Depending on the rules, indices *q*, *r*, *s* may not represent the exact position of punctuations or conjunctions, but their previous or succeeding positions.

construction of dependency relations for a coordinate structure. This leads to an additional index associated with constituents, but in most cases, it is constrained by the position of punctuations and conjunctions.

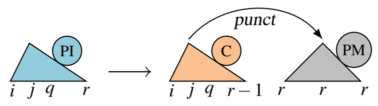
Let us introduce constituent CONJ (or C for short) to represent a conjunct. The first rule for CONJ simply groups the left and right dependents for a conjunct, as follows.

(7) 

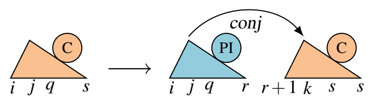
$O(n^3)$

As we explain shortly, CONJ is also used to represent a partially-built coordinate structure. The index k inside the span is used for maintaining the end of the first (head) conjunct, such that the similarity between the first and the subsequent conjuncts can be computed. However, in the above rule, the end position of the span is equal to the end of the conjunct; thus, k appears twice on the left-hand side (lhs) of the rule.

Rules (8)–(9) below deal with a series of conjuncts separated by PUNCTMARK, e.g., "*A*, *B*, *C*, ..." The new constituent, PUNCTINCOMPLETE (PI), represents the sequence CONJ PUNCTMARK.

(8) 

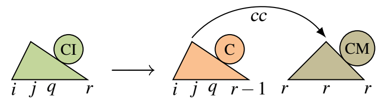
$O(n^2p^2)$

(9) 

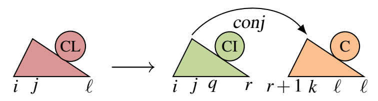
$O(n^3p^3)$

In these rules, q, the index for the end of the first conjunct[3] is inherited by their lhs, as mentioned earlier. With q at our disposal, we can compute the score for the derivation by (9), by taking into account the similarity between the first conjunct on the span (i,q) with head j, and the subsequent conjunct on $(r+1,s)$ with head k.

The following two rules terminate a coordinate structure, when CCMARK and the last CONJ are encountered. We introduce new constituents CCINCOMPLETE (CI) and COMPLETE (CL).

(10) 

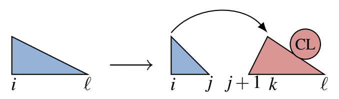
$O(n^2p^2)$

(11) 

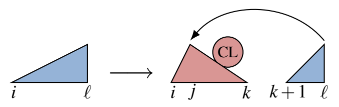
$O(n^4p^2)$

4.3 Interfacing complete coordination with outer parse trees

Rules (12)–(13) connect a completed coordination COMPLETE to the structures made with the standard Eisner-Satta rules.

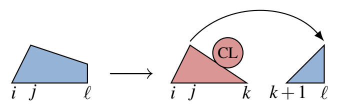
(12) 

$O(n^4)$

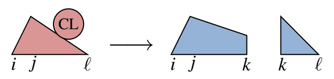
(13) 

$O(n^4)$

We also need rules to handle the case where COMPLETE takes a modifier.

(14) 

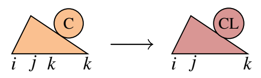
$O(n^4)$

(15) 

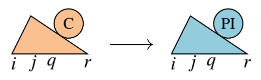
$O(n^4)$

The lhs of rule (14) has a new constituent, which can be regarded as a concatenation of COMPLETELEFT and INCOMPLETERIGHT in Eisner and Satta's rules. Notice also that although the lhs of rule (15) is represented by COMPLETE, it simply means a constituent having both left and right dependents, and loses the meaning of a coordination as a whole.

Furthermore, to deal with recursive coordinations, such as "(*A* & *B*) & *C*," we allow COMPLETE to be CONJ again.[4]

(16) 

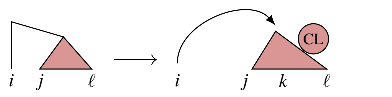
PUNCTINCOMPLETE is also allowed to be CONJ, to tolerate patterns like ", and" (note the comma preceding "and").

(17) 

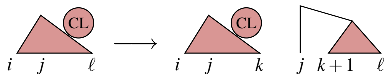
[3]Note that in rules (8)–(11), the index for the end of the first conjunct is denoted by q, meaning that it can take only $O(p)$ different values. This opposes rule (7) in which it was denoted by k, whose range is $O(n)$. This change owes to the fact that the end of the first conjunct is constrained by rules (8) or (10), which require the first CONJ to be followed by a PUNCTMARK or CCMARK; thus the end index for the first conjunct has only $O(p)$ possibilities.

[4]The computational complexity for unary rules is not shown, as it is assumed that their right-hand side is expanded and binarized before application of these rules, in which case the complexity is the same as those of the expanded rules.

There are also cases in which COMPLETE depends upon another COMPLETE, which occurs with a construct such as "(V_1 & V_2)(N_1 & N_2)," where V_i and N_i represent verbs and their objects (nouns), respectively.

(18)


$O(n^4)$

(19)


$O(n^4)$

Here, we used the "hook trick" (Eisner and Satta, 1999; Huang et al., 2005) to reduce the incurred time complexity from $O(n^5)$ to $O(n^4)$.

4.4 Time complexity

If we neglect the number p of punctuations and conjunctions in a sentence, parsing a sentence of length n requires $O(n^4)$ time, which is the same as that of Hara et al.'s (2009) coordination analysis method. For a sentence not containing coordinate conjunctions, the run time is $O(n^3)$.

4.5 A note on spuriousity

Rule (17) causes spurious ambiguity when there are exactly two conjuncts and a comma precedes a coordinate conjunction; e.g., "A, and B." In this case, the dependency between A and the comma can be made with not only rule (17), but also with the standard Eisner-Satta rules. Rule (16) also causes spuriousity in a similar situation. However, these spuriousities can be easily removed by imposing restrictions on rules (7) and (16), such that they are not applicable if $w(k)$ is a comma or semicolon. In the experiment in the next section, these restrictions are used.

5 Coverage of coordination rules

5.1 Experimental setup

We converted the WSJ part of PTB into Stanford basic dependencies[5], and verified the coverage of the proposed rules. Of the 43,948 sentences in WSJ sections 2–23, dependency arcs labeled *conj*, which indicate the presence of coordination, were contained in 40%, or 17,695 sentences.

Dependency structure for coordinations can also be derived with the standard Eisner-Satta rules, but we are only interested in the coverage

[5]we used the Stanford parser (converter) v. 3.5.2, with `-basic` and `-conllx` options to generate gold dependencies.

Count	Type	Description
7	R	multi-word *cc*
5	R	non-projective dependency
5	A	multiple *conj* arcs following *cc*
4	A	head of *punct* is not the first conjunct
3	A	*conj* on the left side of the head
1	R	*conj*s not separated by *cc* or *punct*
1	R	parenthesis (arc labeled *parataxis*)

Table 1: Causes of non-derivable dependencies in WSJ Section 22. We classified these into two types: Error type A is due to errors/inconsistencies in the gold annotation, and type R can be attributed to the deficiency for the proposed rules.

of the proposed rules in WSJ. Hence we treated *conj*-labeled arcs as only derivable with the proposed rules, but not with Eisner-Satta. We thus essentially made Eisner and Satta's rules to fail to derive dependencies on all the sentences containing *conj*-dependencies, and evaluated the coverage of the proposed rules over these sentences. Except for this constraint on *conj*, we ignored all the dependency labels.

5.2 Result

Of the 17,695 sentences containing *conj*-labeled dependencies in WSJ sections 2–23, the proposed rules were able to derive correct dependencies for 94%, or 16,659 sentences.

Table 1 lists the types of failures of our parsing rules and their frequencies in WSJ section 22. About half of the failures were due to inconsistencies or errors in the gold annotations.

6 Conclusion

In this paper, we have proposed a set of dependency parsing rules specifically designed to handle coordination, to be used in combination with Eisner and Satta's rules. The new rules enable the parse scoring function to incorporate the syntactic and semantic similarity of conjuncts. While we are yet to implement such a scoring function, we analyzed the time complexity of the proposed rules and their coverage in the Penn Treebank.

Acknowledgments

This work was supported by JSPS Kakenhi Grant Nos. 24500193 (KH), 15H02749 (MS), and 26240035 (YM).

References

Marie-Catherine de Marneffe and Christopher D. Manning. 2008. Stanford typed dependencies manual. http://nlp.stanford.edu/software/dependencies_manual.pdf. Revised in April 2015.

Jason Eisner and Giorgio Satta. 1999. Efficient parsing for bilexical context-free grammars and head automaton grammars. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL '99)*, pages 457–464, College Park, Maryland, USA.

Atsushi Hanamoto, Takuya Matsuzaki, and Jun'ichi Tsujii. 2012. Coordination structure analysis using dual decomposition. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics (EACL '12)*, pages 430–438, Avignon, France.

Kazuo Hara and Masashi Shimbo. 2007. A discriminative learning model for coordinate conjunctions. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL '07)*, pages 610–619, Prague, Czech Republic.

Kazuo Hara, Masashi Shimbo, Hideharu Okuma, and Yuji Matsumoto. 2009. Coordinate structure analysis with global structural constraints and alignment-based local features. In *Proceedings of the 47th Annual Meeting of the ACL and the 4th IJCNLP of the AFNLP (ACL-IJCNLP '09)*, pages 967–975, Singapore.

Liang Huang, Hao Zhang, and Daniel Gildea. 2005. Machine translation as lexicalized parsing with hooks. In *Proceedings of the 9th International Workshop on Parsing Technology (IWPT '05)*, pages 65–73, Vancouver, British Columbia, Canada.

Mark Johnson. 2007. Transforming projective bilexical dependency grammars into efficiently-parsable CFGs with unfold-fold. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics (ACL '07)*, pages 168–175, Prague, Czech Republic.

Terry Koo and Michael Collins. 2010. Efficient third-order dependency parsers. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1–11, Uppsala, Sweden.

Sadao Kurohashi and Makoto Nagao. 1994. A syntactic analysis method of long Japanese sentences based on the detection of conjunctive structures. *Computational Linguistics*, 20(4):507–534.

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.

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 '10)*, pages 833–841, Beijing, China.

MSTParser Model Interpolation for Multi-source Delexicalized Transfer

Rudolf Rosa and **Zdeněk Žabokrtský**
Charles University in Prague, Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics
Malostranské náměstí 25, Prague, Czech Republic
{rosa, zabokrtsky}@ufal.mff.cuni.cz

Abstract

We introduce interpolation of trained MSTParser models as a resource combination method for multi-source delexicalized parser transfer. We present both an unweighted method, as well as a variant in which each source model is weighted by the similarity of the source language to the target language. Evaluation on the HamleDT treebank collection shows that the weighted model interpolation performs comparably to weighted parse tree combination method, while being computationally much less demanding.

1 Introduction

The task of delexicalized dependency parser transfer (or delex transfer for short) is to train a parser on a treebank for a source language (*src*), using only non-lexical features, most notably part-of-speech (POS) tags, and to apply that parser to POS-tagged sentences of a target language (*tgt*) to obtain dependency parse trees. Delex transfer yields worse results than a supervised lexicalized parser trained on the *tgt* language treebank. However, for languages with no treebanks available, it may be useful to obtain at least a lower-quality parse tree for tasks such as information retrieval.

Usually, multiple *src* treebanks are available, and it is non-trivial to select the best one for a given *tgt* language. Therefore, information from some or all *src* treebanks is usually combined together. The standard ways are to train a parser on the concatenation of all *src* treebanks, or to train a separate parser on each *src* treebank and to combine the parse trees produced by the parsers using a maximum spanning tree algorithm. The tree combination method typically performs better; it can also be easily extended by weighting the *src* parser predictions by similarity of the *src* language to the *tgt* language, which can further improve its results.

In this work, we present a novel method for *src* information combination, based on interpolation of trained parser models. Our approach was motivated by an intuition that the more fine-grained information provided by the *src* edge scores could be of benefit, probably serving as *src* parser confidence. Moreover, model interpolation is significantly less computationally demanding at inference than the parse tree combination method, as instead of running a set of separate *src* parsers, only one parser is run.

2 Related Work

Delex transfer was conceived by Zeman and Resnik (2008), who also introduced two important preprocessing steps – mapping treebank-specific POS tagsets to a common set using Interset (Zeman, 2008), and harmonizing treebank annotation styles into a common style, which later led to the HamleDT harmonized treebank collection (Zeman et al., 2012).

McDonald et al. (2011) applied delex transfer in a setting with multiple *src* treebanks available, finding that the problem of selecting the best *src* treebank without access to a *tgt* language treebank for evaluation is non-trivial, and proposed the treebank concatenation method as a solution. Søgaard and Wulff (2012) introduced weighting into the method, using a POS n-gram model trained on a *tgt* POS-tagged corpus to weight *src* sentences in a weighted perceptron learning scenario (Cavallanti et al., 2010); due to its large computational complexity, we only compare to the unweighted variant in our paper.

The parse tree combination method was introduced by Sagae and Lavie (2006) for a supervised monolingual setting, optionally weighting each *src* parser with a weight based on its accuracy. In (Rosa and Žabokrtský, 2015), we ported the method to a crosslingual setting by combining delex parsers for different languages, weighted by *src-tgt* language similarity; we largely build upon that work in this paper.

Other possibilities of estimating *src-tgt* language similarity for delex transfer include employment of WALS (Dryer and Haspelmath, 2013), focusing e.g. on genealogy distance and word-order features, as done by Naseem et al. (2012) and Täckström et al. (2013), among others.

We are not aware of any prior work on interpolating dependency parser models. However, there is work on interpolating trained phrase-structure parsers, both in a monolingual setting for domain adaptation by McClosky et al. (2010), as well as in a multilingual setting by Cohen et al. (2011).

3 Method

In this section, we present our suggested approach of combining information from multiple *src* treebanks for parsing *tgt* language sentences in a crosslingual delex transfer scenario. The method proceeds as follows:

1. Train a delex parser model on each *src* treebank (Section 3.1).
2. Normalize the parser models (Section 3.2).
3. Interpolate the parser models (Section 3.3).
4. Parse the *tgt* text with a delex parser using the interpolated model.

3.1 Delexicalized MSTParser

Throughout this work, we use MSTperl (Rosa, 2015b), an unlabelled first-order non-projective single-best implementation of the MSTParser of McDonald et al. (2005b), trained using 3 iterations of MIRA (Crammer and Singer, 2003).

The MSTParser model uses a set of binary features F that are assigned weights w_f by training on a treebank. When parsing a sentence, the parser constructs a complete weighted directed graph over the tokens of the input sentence, and assigns each edge e a score s_e which is the sum of weights of features that are active for that edge:

$$s_e = \sum_{\forall f \in F} f(e) \cdot w_f \,. \tag{1}$$

The sentence parse tree is the maximum spanning tree over that graph, found using the algorithm of Chu and Liu (1965) and Edmonds (1967).

The delex feature set we use is based on the set of McDonald et al. (2005a) with lexical features removed. It consists of combinations of signed edge length (distance of head and parent, bucketed for values above 4 and for values above 10) with POS tag of the head, dependent, their neighbours, and all nodes between them. We use the Universal POS Tagset (UPOS) of Petrov et al. (2012). The parser configuration files containing the full feature set, together with the scripts we used for our experiments, are available in (Rosa, 2015a).

3.2 Model Normalization

An important preliminary step to model interpolation is to normalize each of the trained models, as the feature weights in models trained over different treebanks are often not on the same scale (we do not perform any regularization during the parser training). We use a simplified version of normalization by standard deviation. First, we compute the uncorrected sample standard deviation of the weights of the features in the model as

$$s_M = \sqrt{\frac{1}{|M|} \sum_{\forall f \in M} (w_f - \bar{w})^2} \,, \tag{2}$$

where $\bar{w}$ is the average feature weight, and $|M|$ is the number of feature weights in model M; only features that were assigned a weight by the training algorithm are taken into account.

We then divide each feature weight by the standard deviation:[1]

$$\forall f \in M : w_f := \frac{w_f}{s_M} \,. \tag{3}$$

The choice of normalization by standard deviation is based on its high and stable performance on our development set, and Occam's razor.[2]

3.3 Model Interpolation

The interpolated model is a linear combination of the normalized models trained over the *src* treebanks. The result is a model that can be used in the same way as a standard MSTParser model.

[1]We have not found any further gains in performance when subtracting the sample mean from the weight before the division; the MSTParser models seem to be typically centered very similarly.

[2]We tried 12 normalization schemes, nearly all of which achieved an improvement of 2.5% to 5% UAS absolute over an interpolation of unnormalized models on average, but often with large differences for individual languages. Another well-performing method was to divide each feature weight by the sum of absolute values of all feature weights in the model; or a similar method, applied during inference individually for each sentence, using only the feature weights that fired for the sentence to compute the divisor.

In unweighted model interpolation, the weight of each feature (w_f) is computed as the sum of the weights of that feature in the *src* models ($w_{f,src}$):

$$\forall f \in F : w_f = \sum_{\forall src} w_{f,src} \,. \tag{4}$$

In the weighted variant of model interpolation, we extend (4) with multiplication by the $KL^{-4}_{cpos^3}$ weight of Rosa and Žabokrtský (2015):

$$\forall f \in F : \\ w_f = \sum_{\forall src} w_{f,src} \cdot KL^{-4}_{cpos^3}(tgt, src) \,. \tag{5}$$

The $KL^{-4}_{cpos^3}$ weight corresponds to the similarity of the *src* language to the *tgt* language, and is defined as the negative fourth power of the KL divergence (Kullback and Leibler, 1951) of coarse POS tag trigram distributions in *tgt* and *src* corpora:

$$KL^{-4}_{cpos^3}(tgt, src) = \\ \left(\sum_{\substack{\forall cpos^3 \\ \in tgt}} f_{tgt}(cpos^3) \cdot \log \frac{f_{tgt}(cpos^3)}{f_{src}(cpos^3)} \right)^{-4} , \tag{6}$$

where $cpos^3$ is a UPOS trigram, and $f(cpos^3)$ is its relative frequency in a *src* or *tgt* corpus.[3]

4 Baseline Methods

In this section, we describe the two baseline resource combination methods against which we compare our model interpolation method.

4.1 Treebank Concatenation

The treebank concatenation method of McDonald et al. (2011) proceeds as follows:

1. Concatenate all *src* treebanks.
2. Train a delex parser on the resulting treebank.
3. Apply the parser to the *tgt* text.

4.2 Parse Tree Combination

The parse tree combination method is defined by Rosa and Žabokrtský (2015) in the following way:

1. Train a delex parser on each *src* treebank.
2. Apply each of the parsers to the *tgt* sentence, obtaining a set of parse trees.
3. Construct a weighted directed graph over *tgt* sentence tokens, with each edge assigned a score equal to the number of parse trees that contain this edge. (i.e., each parse tree contributes by 0 or 1 to the edge score). In the weighted variant, the contribution of each *src* parse tree is multiplied by $KL^{-4}_{cpos^3}$ (*tgt*, *src*).
4. Find the maximum spanning tree over the graph with the Chu-Liu-Edmonds algorithm.

[3] $f_{src}(cpos^3) := \frac{1}{N}$ if the *src* corpus does not contain the given trigram (N is the number of tokens in the corpus).

Note that if each *src* parse tree contributed with a (normalized) score of the edge as assigned by its model rather than with a 0 or 1, this method would be equivalent to the model interpolation method.

5 Dataset

We carry out all experiments using HamleDT 2.0 (Rosa et al., 2014), a collection of 30 treebanks converted into Universal Stanford Dependencies (de Marneffe et al., 2014). We use gold-standard UPOS tags in all experiments; while this is not fully realistic in the setting of under-resourced languages, there exist high-performance semi-supervised taggers that could be used instead of gold tags (Das and Petrov, 2011; Agić et al., 2015), which we plan to evaluate in future. We use the treebank training sections for parser training and $KL^{-4}_{cpos^3}$ computation, and the test sections for evaluation. We used 12 of the treebanks as a development set to select the model normalization method to avoid overfitting it to the dataset.[4]

6 Evaluation

Table 1 contains the results of our model interpolation methods, as well as the baseline methods. For each *tgt* language, all remaining 29 *src* treebanks were used for parser training. We base our evaluation on comparing absolute differences in UAS on the whole set of 30 languages as targets.[5]

The performance of the weighted model interpolation is comparable to the weighted tree combination – the difference in average UAS of the methods is lower than 0.1%, with model interpolation achieving a higher UAS than the tree combination for 16 of the 30 *tgt* languages. This shows

[4] The development set was chosen to contain multiple members of several language families (Uralic, Romance), as well as a very solitary language (Japanese), etc.; also, we cared that both smaller and larger treebanks are represented.

[5] The results of our method are generally better on the test set than on the development set, suggesting that no overfitting happened.

Target	Unweighted			Weighted	
language	Conc	Tree	Inter	Tree	Inter
Bengali	61.0	63.2	**67.1**	66.7	**66.9**
Czech	**60.5**	60.4	57.5	**65.8**	65.2
Danish	**56.2**	54.4	48.9	**50.3**	49.5
German	12.6	**27.6**	18.2	56.8	**61.6**
English	12.3	**21.1**	16.2	42.6	**48.6**
Basque	**41.2**	40.8	39.5	30.6	**34.9**
Anc. Greek	43.2	**44.7**	41.4	42.6	**44.0**
Latin	38.1	**40.3**	39.7	**39.7**	39.5
Dutch	55.0	**56.2**	54.2	58.7	**59.4**
Portuguese	62.8	**67.2**	62.8	62.7	**63.7**
Romanian	44.2	**51.2**	48.6	50.0	**50.3**
Russian	55.5	**57.8**	53.3	**57.2**	56.3
Slovak	52.2	**59.6**	55.7	58.4	**60.6**
Slovenian	45.9	**47.1**	42.8	**53.9**	49.6
Swedish	45.4	**52.3**	49.4	**50.8**	50.4
Tamil	27.9	**28.0**	27.6	**40.0**	37.3
Telugu	67.8	68.7	**72.9**	**77.4**	**77.4**
Turkish	18.8	23.2	**25.3**	**41.1**	34.8
Average	44.5	**48.0**	45.6	52.5	**52.8**
Std. dev.	16.9	**15.0**	16.0	**11.8**	12.0
Arabic	**37.0**	35.3	30.7	**41.3**	34.6
Bulgarian	64.4	**66.0**	60.3	67.4	**68.5**
Catalan	56.3	**61.5**	58.5	72.4	**72.4**
Greek	**63.1**	62.3	59.6	63.8	**64.1**
Spanish	59.9	**64.3**	60.4	72.7	**72.7**
Estonian	67.5	**70.5**	67.4	**72.0**	71.7
Persian	30.9	**32.5**	29.5	**33.3**	28.6
Finnish	**41.9**	41.7	41.5	**47.1**	44.7
Hindi	24.1	24.6	**26.2**	27.2	**32.7**
Hungarian	55.1	56.5	**57.4**	51.2	**53.0**
Italian	52.5	**59.5**	56.0	59.6	**60.1**
Japanese	**29.2**	28.8	27.2	**34.1**	33.0
Average	48.5	**50.3**	47.9	**53.5**	53.0
Std. dev.	**15.2**	16.5	15.6	**16.7**	17.4
Average	46.1	**48.9**	46.5	**52.9**	52.9
Std. dev.	16.1	**15.4**	15.6	**13.7**	14.1

Table 1: UAS on test *tgt* treebanks (upper part of table) and development *tgt* treebanks (lower part).
Conc = Treebank concatenation
Tree = Parse tree combination
Inter = Model interpolation
Average = Average UAS (on test/development/all)
Std. dev. = Standard sample deviation of UAS, serving as an indication of robustness of the method

that weighted model interpolation is a good alternative to weighted tree combination.

In the unweighted setting, the situation is quite different, with model interpolation scoring much lower than tree combination (-2.4%), and only slightly higher than treebank concatenation (+0.4%) on average. This suggests that, contrary to our original intuition, edge scores assigned by the *src* models are not a good proxy for parser confidence, not even when appropriately normalized.[6] Furthermore, the weighted methods generally outperform the unweighted ones (by +4.0% for tree combination and by +6.4% for model interpolation on average), which suggests, among other, that the *src*-*tgt* language similarity is much more important than the exact values of *src* edge scores for resource combination in delex transfer.

7 Conclusion

We presented trained parser model interpolation as an alternative method for multi-source crosslingual delexicalized dependency parser transfer. Evaluation on a large collection of treebanks showed that in a setting where the source languages are weighted by their similarity to the target language, model interpolation performs comparably to the parse tree combination approach. Moreover, model interpolation is significantly less computationally demanding than the tree combination when parsing the target text, as the interpolation can be efficiently performed beforehand, thus only requiring to invoke a single parser at runtime, while in the tree combination approach, each source parser has to be invoked individually.

In the unweighted setting, model interpolation consistently performed much worse than tree combination, which we find rather surprising, and we therefore plan to further investigate this in future. Still, the weighted methods generally outperformed the unweighted ones, and as the language similarity measure that we used only requires the source treebanks and a target POS-tagged text, i.e. exactly the resources that are required even for the unweighted delex transfer methods, there is little reason not to employ the weighting. Therefore, the low performance of the unweighted model interpolation is of less importance than its high performance in the weighted setting.

In this work, we only used the unlabelled MST-Parser for all experiments. We believe that extending our method to other parsers constitutes an interesting path for future research.

Acknowledgments

This research was supported by the grants GAUK 1572314, SVV 260 224 and FP7-ICT-2013-10-610516 (QTLeap). This work has been using language resources developed, stored and distributed by the LINDAT/CLARIN project of the Ministry of Education, Youth and Sports of the Czech Republic (project LM2010013).

[6]The same tendency was observed across all normalization methods evaluated on the development set.

References

Željko Agić, Dirk Hovy, and Anders Søgaard. 2015. If all you have is a bit of the bible: Learning POS taggers for truly low-resource languages. In *Proceedings of ACL-IJCNLP*. Hrvatska znanstvena bibliografija i MZOS-Svibor.

Giovanni Cavallanti, Nicolo Cesa-Bianchi, and Claudio Gentile. 2010. Linear algorithms for online multitask classification. *The Journal of Machine Learning Research*, 11:2901–2934.

Yoeng-Jin Chu and Tseng-Hong Liu. 1965. On shortest arborescence of a directed graph. *Scientia Sinica*, 14(10):1396.

Shay B. Cohen, Dipanjan Das, and Noah A. Smith. 2011. Unsupervised structure prediction with non-parallel multilingual guidance. In *Proceedings of EMNLP*, pages 50–61, Stroudsburg, PA, USA. ACL.

Koby Crammer and Yoram Singer. 2003. Ultraconservative online algorithms for multiclass problems. *The Journal of Machine Learning Research*, 3:951–991.

Dipanjan Das and Slav Petrov. 2011. Unsupervised part-of-speech tagging with bilingual graph-based projections. In *Proceedings of ACL-HLT*, pages 600–609. ACL.

Marie-Catherine de Marneffe, Natalia Silveira, Timothy Dozat, Katri Haverinen, Filip Ginter, Joakim Nivre, and Christopher D. Manning. 2014. Universal Stanford dependencies: A cross-linguistic typology. In *Proceedings of LREC'14*, Reykjavík, Iceland. ELRA.

Matthew S. Dryer and Martin Haspelmath, editors. 2013. *WALS Online*. Max Planck Institute for Evolutionary Anthropology, Leipzig.

Jack Edmonds. 1967. Optimum branchings. *Journal of Research of the National Bureau of Standards B*, 71(4):233–240.

Solomon Kullback and Richard A Leibler. 1951. On information and sufficiency. *The annals of mathematical statistics*, pages 79–86.

David McClosky, Eugene Charniak, and Mark Johnson. 2010. Automatic domain adaptation for parsing. In *Proceedings of HLT-NAACL*, pages 28–36. ACL.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005a. Online large-margin training of dependency parsers. In *Proceedings of ACL*, pages 91–98. ACL.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005b. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of HLT-EMNLP*, pages 523–530. ACL.

Ryan McDonald, Slav Petrov, and Keith Hall. 2011. Multi-source transfer of delexicalized dependency parsers. In *Proceedings of EMNLP*, pages 62–72, Stroudsburg, PA, USA. ACL.

Tahira Naseem, Regina Barzilay, and Amir Globerson. 2012. Selective sharing for multilingual dependency parsing. In *Proceedings of ACL*, pages 629–637, Stroudsburg, PA, USA. ACL.

Slav Petrov, Dipanjan Das, and Ryan McDonald. 2012. A universal part-of-speech tagset. In *Proc. of LREC-2012*, pages 2089–2096, Istanbul, Turkey. ELRA.

Rudolf Rosa and Zdeněk Žabokrtský. 2015. KL_{cpos^3} – a language similarity measure for delexicalized parser transfer. In *Proceedings of ACL-IJCNL*, Stroudsburg, PA, USA. ACL.

Rudolf Rosa, Jan Mašek, David Mareček, Martin Popel, Daniel Zeman, and Zdeněk Žabokrtský. 2014. HamleDT 2.0: Thirty dependency treebanks stanfordized. In *Proceedings of LREC 2014*, pages 2334–2341, Reykjavík, Iceland. ELRA.

Rudolf Rosa. 2015a. MSTperl delexicalized parser transfer scripts and configuration files. `http://hdl.handle.net/11234/1-1485`.

Rudolf Rosa. 2015b. MSTperl parser (2015-05-19). `http://hdl.handle.net/11234/1-1480`.

Kenji Sagae and Alon Lavie. 2006. Parser combination by reparsing. In *Proceedings of HLT-NAACL*, pages 129–132. ACL.

Anders Søgaard and Julie Wulff. 2012. An empirical etudy of non-lexical extensions to delexicalized transfer. In *COLING (Posters)*, pages 1181–1190.

Oscar Täckström, Ryan McDonald, and Joakim Nivre. 2013. Target language adaptation of discriminative transfer parsers. In *Proceedings of HLT-NAACL*, pages 1061–1071.

Daniel Zeman and Philip Resnik. 2008. Cross-language parser adaptation between related languages. In *IJCNLP 2008 Workshop on NLP for Less Privileged Languages*, pages 35–42, Hyderabad, India. Asian Federation of Natural Language Processing, International Institute of Information Technology.

Daniel Zeman, David Mareček, Martin Popel, Loganathan Ramasamy, Jan Štěpánek, Zdeněk Žabokrtský, and Jan Hajič. 2012. HamleDT: To parse or not to parse? In *Proceedings of LREC'12*, Istanbul, Turkey, May. ELRA.

Daniel Zeman. 2008. Reusable tagset conversion using tagset drivers. In *Proceedings of LREC 2008*, pages 213–218, Marrakech, Morocco. ELRA.

Non-Deterministic Oracles for Unrestricted Non-Projective Transition-Based Dependency Parsing

Anders Björkelund
University of Stuttgart
Institute for Natural Language Processing
Stuttgart, Germany
anders@ims.uni-stuttgart.de

Joakim Nivre
Uppsala University
Department of Linguistics and Philology
Uppsala, Sweden
joakim.nivre@lingfil.uu.se

Abstract

We study non-deterministic oracles for training non-projective beam search parsers with swap transitions. We map out the spurious ambiguities of the transition system and present two non-deterministic oracles as well as a static oracle that minimizes the number of swaps. An evaluation on 10 treebanks reveals that the difference between static and non-deterministic oracles is generally insignificant for beam search parsers but that non-deterministic oracles can improve the accuracy of greedy parsers that use swap transitions.

1 Introduction

Training transition-based dependency parsers relies on an **oracle** – a function that, given a parser configuration and a gold dependency tree, returns the correct transition. The sequence of transitions required to derive a given dependency tree is, however, typically not unique, and for certain configurations more than one transition is correct. This issue has typically been dealt with by defining a canonical order among the transitions, thereby resolving such ambiguities in a deterministic way. In addition to the determinism, standard oracles also make the assumption that the gold tree can be recovered from the current configuration. Oracles with this behavior are known as **static** oracles.

Recently, much work has been devoted to the development of *dynamic* oracles that do away with both of these assumptions (Goldberg and Nivre, 2013; Goldberg et al., 2014; Gómez-Rodríguez et al., 2014). Dynamic oracles have been shown to be very successful for training greedy parsers. Since greedy parsers typically suffer from error propagation, dynamic oracles enable the parsers to learn to do “the next best thing” after having made a mistake, resulting in considerable improvements in parsing accuracy.

Nevertheless, greedy transition-based parsers still lag behind search-based parsers that explore a larger set of possible transition sequences. Search-based parsers are typically realized through beam search and trained using global learning, where a discriminative model is trained to score not just single transitions, but a sequence of transitions (Zhang and Clark, 2008). The combination of non-greedy inference and global learning enables search-based parsers to overcome the error propagation problem. However, since the model is globally trained, oracles that can recover from past mistakes and do the next best thing are not applicable in this scenario. On the other hand, it is an open question whether search-based parsers should be trained using static oracles, or whether their performance can be further increased by using a **non-deterministic** oracle, i.e., an oracle that considers all possible transition sequences that can derive the gold dependency tree.

We evaluate the hypothesis that transition-based parsers with beam search can be improved by using non-deterministic oracles during training. We do this in the context of the transition system by Nivre (2009), henceforth SwapStandard, which extends the ArcStandard system (Nivre, 2004) with a swap transition to accommodate non-projective dependency trees. This system has been shown to be very effective with beam search, even rivaling graph-based dependency parsers (Bohnet and Nivre, 2012; Bohnet et al., 2013). Empirically, we find that the utility of non-deterministic oracles for training beam search parsers is rather limited and typically the difference compared to a static oracle is insignificant. However, our experiments also show that the non-deterministic oracles can be beneficial when training greedy parsers, a result that has not previously been shown for non-projective systems.

The main contribution of this paper is the first characterization of a non-deterministic oracle for the SwapStandard system, based on a thorough analysis of spurious ambiguities. As a side-result, we also arrive at a static oracle that minimizes the number of swap transitions required to parse any non-projective dependency tree, thereby solving a previously open problem (Nivre, 2009; Nivre et al., 2009). In addition, we provide the first empirical evaluation of non-deterministic oracles for training beam search parsers, as well as the first evaluation with greedy parsers using a non-projective transition system.

2 Related Work

During the last decade, a plethora of transition systems has been described. Early systems, such as ArcEager (Nivre, 2003) and ArcStandard (Nivre, 2004) were restricted to projective structures. Several systems that can accommodate non-projective structures have subsequently been described (Attardi, 2006; Gómez-Rodríguez and Nivre, 2010, *inter alia*). These systems are, however, restricted to certain subsets of non-projective structures. In contrast, SwapStandard imposes no such restrictions and is able to parse unrestricted non-projective structures.

Dynamic oracles were first introduced by Goldberg and Nivre (2012) for the ArcEager system. They also proposed the standard way of exploiting dynamic oracles for training greedy parsers known as *training with exploration*. Here, the idea is that sometimes erroneous transitions are predicted during training. The dynamic oracle then comes into play by guiding the model towards the best possible tree, subject to the mistakes that have already been made. However, search-based parsers model the parsing problem as a structured prediction problem and are trained to predict optimal sequences of transitions for an entire sentence. Training with exploration is thus not applicable.

More recent work on dynamic oracles has primarily focused on developing dynamic oracles for other transition systems. Goldberg et al. (2014) present dynamic oracles for the ArcStandard system and the LR-spine parser by Sartorio et al. (2013). The only dynamic oracle for non-projective dependency trees was introduced by Gómez-Rodríguez et al. (2014) for a special case of Attardi's (2006) system. To date no dynamic oracle has been presented for transition systems that can handle unrestricted non-projective dependencies.

The underlying idea in our work is that there may be more than a single decomposition (i.e., transition sequence) that can recover the correct output (dependency tree). Rather than selecting a single unique such decomposition, the choice of decomposition can be thought of as **latent** and deferred to the machine learning algorithm. This approach has been shown to be successful for a number of tasks, including coreference resolution (Fernandes et al., 2012), semantic parsing (Zhou et al., 2013), and statistical machine translation (Yu et al., 2013), to name a few.

3 Transition System

We begin by describing our notation and the SwapStandard system. For simplicity we omit the inclusion of arc labels from this description, although for the experimental evaluation we implement a labeled version of this system. For a more formal description of the system, as well as proofs of soundness and completeness, we refer the reader to Nivre (2009).

The SwapStandard system operates on *configurations* $c = (\Sigma, B, A)$, where Σ denotes a stack of partially processed tokens, B denotes a buffer of remaining input tokens, and A is a set of arcs. We denote stack items by $s_i, i \geq 0$ where s_0 denotes the topmost item on the stack. Similarly, let $b_i, i \geq 0$ denote the items of the buffer, b_0 denoting the first element on the buffer. Finally, let $h \rightarrow d$ denote an arc from the head h to the dependent d.

The system begins in an *initial configuration* $c_0 = ([0], [1, 2, 3, ...], \emptyset)$, where the stack consists solely of the root token (numbered 0), all input tokens are on the buffer (numbered 1, 2, ...), and the arc set is the empty set. A configuration is *terminal* when the buffer is empty and the stack consists only of the root token 0.

The possible transitions of the system are

- Shift (`SH`) – removes b_0 from the buffer and pushes it onto the stack,
- LeftArc (`LA`) – introduces an arc $s_0 \rightarrow s_1$ and removes s_1 from the stack,
- RightArc (`RA`) – introduces an arc $s_1 \rightarrow s_0$ and removes s_0 from the stack,
- Swap (`SW`) – removes s_1 from the stack and places it as the first element on the buffer.

The `SW` transition reorders tokens from the input on the fly, enabling the system to recover non-projective trees. Informally, a dependency tree is

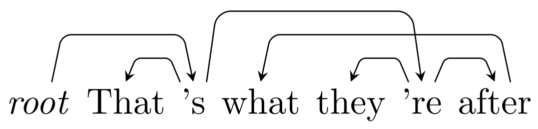


Figure 1: A non-projective dependency tree.

non-projective if it cannot be drawn without any crossing arcs. An example sentence with a non-projective tree is shown in Figure 1.

The total number of transitions required to parse a sentence of length n is bounded from below by $2n$ since every token needs to be shifted onto the stack once and attached to its head through an `LA` or `RA` once. Additionally, every `SW` moves a token back onto the buffer which subsequently needs to be shifted again, adding $2k$ transitions for k `SW`.

Eager oracle. Nivre (2009) presents a static oracle for the SwapStandard system. A high-level algorithmic description of this oracle is shown in Algorithm 1. The functions CANLA, CANRA, and CANSW define the requirements for the corresponding transitions. For `LA` the requirement is that s_0 is the head of s_1 and that s_1 has already collected all of its own dependents, and vice versa for `RA`.

To decide when `SW` can be applied, Nivre (2009) introduces the notion of **projective order** which is obtained through an inorder traversal of the dependency tree. The projective order is a total order over the tokens of a sentence. If the tokens are sorted accordingly, the tree becomes projective. The `SW` transition is allowed when s_0 precedes s_1 according to the projective order.

For a given sentence x, which is understood to include a representation of x's dependency tree and projective order, Algorithm 1 can be used to create a sequence of transitions that can derive the corresponding dependency tree. Specifically, iteratively call the oracle starting from x's initial configuration until the terminal configuration has been reached. This transition sequence is the oracle sequence for x.

In the example from Figure 1, the projective order is *That* $<$ *'s* $<$ *they* $<$ *'re* $<$ *what* $<$ *after*. The difference compared to the original word order is that *what* has been moved two tokens to the right. This means that `SW` is permissible when either of *they* or *'re* are s_0 and *what* is s_1. The oracle would apply `SW` when *what* is s_1 and *they* is s_0. It would then continue with two more `SH` followed by another `SW`, thereby obtaining the projective order. Since this oracle applies `SW` whenever the projective order admits it, we refer to it as EAGER.

Algorithm 1 Generic static oracle

```
Input: Configuration $c$, sentence $x$
1: if CANLA($c, x$) then
2:     return LA
3: else if CANRA($c, x$) then
4:     return RA
5: else if CANSW($c, x$) then
6:     return SW
7: else
8:     return SH
```

Lazy oracle. For non-projective trees, the sequence given by EAGER is often not the shortest sequence, and shorter sequences that require fewer `SW` transitions to produce the same parse may be possible. Drawing upon this observation, Nivre et al. (2009) present an improved oracle that considerably reduces the number of swaps. Their oracle is based on the same basic structure as the one in Algorithm 1, but the semantics of CANSW are redefined. The oracle relies on a notion of **maximal projective components** (MPCs). In addition to requiring that s_0 and s_1 appear in the wrong order with respect to the projective order, this oracle also requires s_0 and s_1 not to be part of the same MPCs. MPCs are defined as the resulting subtrees obtained by running the oracle parser without `SW` until it hits a dead end. Nivre et al. (2009) show that this oracle substantially reduces the number of `SW` transitions, sometimes by up to 80%.

In the example from Figure 1 this oracle would not admit the first `SW` transition when *what* is s_1 and *they* is s_0. Instead, it would continue by shifting *'re* and then attaching *they* through an `LA`. As this oracle tries to postpone `SW` transitions when possible, we refer to it as LAZY.

4 Spurious Ambiguities

The static oracles presented above can produce a sequence of transitions to derive any dependency tree. By design, the algorithm selects among the possible transitions using a pre-defined order, i.e., the order of the tests in the if-clauses. This procedure yields a deterministic sequence of transitions for any dependency tree. This sequence is not necessarily the only correct one for a given dependency tree. In fact, most dependency trees seen in standard treebanks can be derived from more than one sequence of transitions. Different such

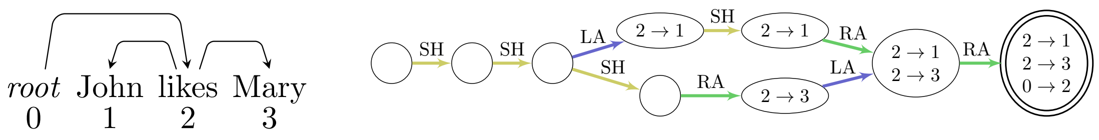


Figure 2: A dependency tree which exhibits the SH-LA ambiguity (left) and a lattice that encodes the two alternative transition sequences (right).

sequences always start and end with the same transitions, however, somewhere along the way a fork point occurs where more than one transition can be applied while still keeping the desired dependency tree reachable. We call these fork points *spurious ambiguities* as they all lead to the same tree. We have already seen one such ambiguity above, comparing EAGER and LAZY, namely the SH-SW ambiguity. Below we review the remaining spurious ambiguities of the SwapStandard system and give examples of each kind.

SH-LA ambiguity. While the canonical way of constructing the dependency tree is to attach left dependents as early as possible, these decisions can sometimes be delayed. Consider the example sentence on the left in Figure 2. Here, the left dependent of *likes* need not be attached before the right. The example thus has two possible transition sequences that create the given dependency tree. These sequences can be illustrated in the form of a lattice, as depicted on the right in Figure 2. Nodes in the lattice correspond to parser configurations and arcs between them to transitions (color-coded for different transitions). The initial configuration is to the left, and the terminal configuration is on the right (indicated by a double circle). The text in the nodes show the arcs that have been constructed thus far.[1] The SH-LA ambiguity gives rise to the fork point where there are two parallel paths, corresponding to early and late attachments of the left dependent. This ambiguity is not specific to SwapStandard, but also occurs in the projective ArcStandard system. In the projective case, this type of ambiguity always arises when the parser can make an LA attachment but b_0 is dominated by s_0.

SH-RA ambiguity. While the SH-RA ambiguity is not possible in the plain ArcStandard system the introduction of SW enables this ambiguity. In the ArcStandard system, applying an SH when an RA

[1]While the nodes only display the arc set so far, the merge points in the lattice truly correspond to equivalent states where the stack, buffer, and arc set are all identical.

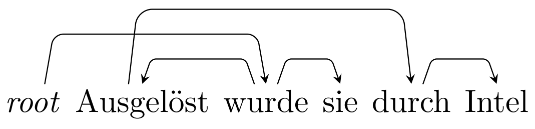


Figure 3: A non-projective dependency tree which exhibits the SH-RA ambiguity.

is possible results in "burying" tokens on the stack such that they are irretrievable. Since the SW transition moves tokens out of the stack and back onto the buffer, it is sometimes possible to recover these buried tokens and thus do the RA at a later point.

Consider the German sentence in Figure 3. The static oracle would parse this sentence by first applying three SH followed by an RA, attaching *sie* to *wurde*. However, the projective order of this sentence is *Ausgelöst* $<$ *durch* $<$ *Intel* $<$ *wurde* $<$ *sie*. The system is thus able to delay the RA transition, do another SH and then swap *wurde* and *sie* past *durch* and handle this attachment later. The lattice in Figure 4 shows all ambiguities for this sentence. The first SH-RA ambiguity is highlighted.

In comparison to the SH-LA ambiguity seen earlier, the SH-RA ambiguity always relies on additional SW transitions and thus leads to longer transition sequences. Note that the same logic also holds for LA – sometimes both the head and dependent of an LA transition can be swapped out, creating a second kind of SH-LA ambiguity in addition to the one seen already.

4.1 Non-deterministic oracles

Recall that the main hypothesis of our work is that a search-based parser can be further improved by using a non-deterministic oracle. So far we have seen three types of spurious ambiguities. It is easy to convince oneself that no other ambiguities are possible – by sheer enumeration of the possible pairs of transitions, any other ambiguities would involve one of the pairs LA-RA, LA-SW, or RA-SW. An LA-RA ambiguity would only be possible if s_0 is the head of s_1 and s_1 is the head of s_0 which implies that the tree has a cycle. The LA-SW and

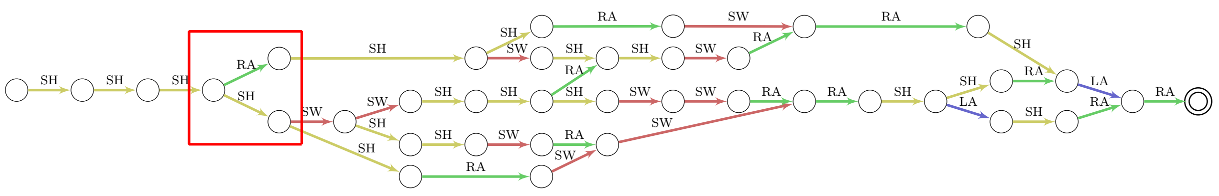


Figure 4: A lattice which encodes all possible transition sequences to parse the sentence from Figure 3. The initial SH-RA ambiguity is highlighted in the red box.

Algorithm 2 Can shift

```
Input: Configuration $c$, sentence $x$
 1: if $|c.B| = 0$ then                    ▷ Not possible if buffer is empty
 2:     return false
 3: $c = $ DoSH($c$)                         ▷ Initial shift
 4: while $\neg$TERMINAL($c$) do
 5:     if CANLA($c, x$) then
 6:         $c = $ DoLA($c$)
 7:     else if CANRA($c, x$) then
 8:         $c = $ DoRA($c$)
 9:     else if CANSW$_{\text{Eager}}$($c, x$) then
10:         $c = $ DoSW($c$)
11:     else if $|c.B| > 0$ then           ▷ SH if buffer is not empty
12:         $c = $ DoSH($c$)
13:     else
14:         return false                     ▷ Hit dead end
15: return true
```

RA-SW ambiguities would swap a dependent past its head, or a head past its dependent. This would, however, violate the projective order and is therefore also not possible.

The key question that needs to be resolved in order to construct a non-deterministic oracle is when an SH transition can be applied. The EAGER oracle defines when the other three transitions can be applied, but treats SH as a fallback when no other transitions are possible. So when is SH applicable? One way to find out is to try an SH and see if there is any way to recover the full parse. This procedure is described in Algorithm 2. The algorithm applies an initial shift (line 3) and uses the EAGER oracle from then on (lines 4 to 14). If the parser can recover the correct parse, then SH is permissible. If not, the parser will eventually end up in a dead end where the buffer is empty, the stack contains several items, but no other transitions are applicable (line 14).[2]

The reason this algorithm works is that after applying the initial SH, it prefers all the other transitions over additional SH transitions. The other transitions are all taking clear steps towards avoiding dead ends, either by introducing arcs (removing tokens from the system) or by applying swaps (permuting the words in the direction of the projective order by moving tokens back from the stack onto the buffer).

The procedure outlined in Algorithm 2 allows us to construct a non-deterministic oracle for the SwapStandard system. Specifically, whenever either of LA, RA, or SW are permissible, the oracle also checks if SH is possible. If neither of LA, RA, or SW are permissible, SH is returned. This oracle allows for all possible ambiguities and we refer to it as ND-ALL.

It could be argued that a parser could profit from having a more eager treatment of arc attachments and that the SH-LA and SH-RA ambiguities should be avoided. As for the SH-SW ambiguity, we have seen that there is a continuum of how eagerly SW transitions should be applied, ranging from EAGER to LAZY (and beyond, as we will see shortly). The static oracles apply SW according to the predefined rules that are primarily grounded in tree structural characteristics. These rules may not be the most motivated from a linguistic perspective, and there could be a more systematic treatment of swaps lying somewhere in between that is easier to learn. In order to investigate whether the parser is able to learn better such patterns latently, we also construct an oracle that only permits the SH-SW ambiguity but no others. We will refer to this oracle as ND-SW.

4.2 Minimally swapping oracle

While the LAZY oracle considerably reduces the number of SW transitions, this oracle still does not always yield the minimal number of swaps for a given dependency tree. Given the possibility to tell when an SH is possible (Algorithm 2), we can construct lattices as those shown above for any dependency tree. By searching this lattice for the shortest path from the initial state to the terminal

[2]The worst case runtime of this algorithm is $O(n^2)$, although it is enough to halt the search when the stack has been reduced to only two tokens (one being root), since from that point on the gold tree has to be recoverable. In practice we observed that applying Algorithm 2 during training had negligible effect on overall training time.

state, the shortest possible transition sequence can be found. As this oracle minimizes the number of SW transitions, we refer to it as MINIMAL.[3]

This procedure cannot be formalized as concisely as Algorithm 1 and relies on searching the corresponding lattice. But it should be noted that when a static oracle is used for training, the transition sequence for each sentence only needs to be computed once before training.

The lattices can grow extremely large, to the point where the lattice of a single sentence cannot be kept in main memory.[4] A depth-first search that keeps a stack of fork points in memory circumvents this problem. After reaching the terminal state the first time, only the path chosen and the number of transitions need to be remembered. Any further paths that have not reached the terminal state after as many transitions as the currently seen shortest path can be terminated immediately. While this oracle is clearly slower than the other static ones, we found that the extra overhead is negligible in comparison to overall training time.

5 Training

We train the parser with a variant of the structured perceptron (Collins, 2002) and use a beam size of 20. We follow Bohnet et al. (2013) and use the the Passive-Aggressive algorithm (Crammer et al., 2006). We deviate slightly from the previous work and use the Max-Violation framework (Huang et al., 2012) rather than early update (Collins and Roark, 2004), as we found that it required fewer iterations and yielded slightly higher scores, both for static and non-deterministic oracles. Following standard practice, we also apply parameter averaging (Collins, 2002).

When training with a static oracle the correct configuration to update against is well-defined, but with a non-deterministic oracle there may be more than one correct configuration and it is unclear against which to update. Yu et al. (2013) suggest to compare with the highest scoring correct configuration at every step but in initial experiments we found that this performed rather poorly. Instead, we apply beam search in a constrained setting to arrive at a single best correct sequence using the current parameters. This sequence is the *latent* gold sequence and is recomputed for every sentence during every iteration.

When we train a greedy parser we fall back to the standard perceptron algorithm using a non-deterministic oracle (Goldberg and Nivre, 2013; Goldberg et al., 2014). Since this model is not globally trained, only the choice of the next transition is latent but the basic principle is the same.

The feature model we use is primarily based on that of Zhang and Nivre (2011) with the obvious adaptations to the SwapStandard setting. Additional features are taken from other recent work on parsers using the SwapStandard system (Bohnet and Nivre, 2012; Bohnet et al., 2013). Following the line of work presented by Bohnet et al. we also replace the feature mapping function by a hash function which enables the use of negative features and yields a considerable speed improvement (Bohnet, 2010).[5]

6 Experiments

In total we experiment with five different oracles. The three static ones, EAGER, LAZY, and MINIMAL, all use a single unique transition sequence for every sentence in the training data. The two non-deterministic oracles, ND-SW and ND-ALL, create latent transition sequences on the fly relying on the current parameters and may change across training iterations. Our main hypothesis is that the latent sequences created by the non-deterministic oracles are easier to learn and generalize better to unseen data, leading to increased accuracy.

Data sets. We evaluate the oracles on ten treebanks. Specifically, we use the nine treebanks from the SPMRL 2014 Shared Task (Seddah et al., 2014), comprising Arabic, Basque, French, German, Hebrew, Hungarian, Korean, Polish, and Swedish. For these treebanks we use the train/dev/test splits provided by the Shared Task organizers. Additionally, we use the English Penn Treebank (Marcus et al., 1993) converted to Stanford dependencies (de Marneffe et al., 2006) with the standard split, sections 2-21 for training, section 24 for development, and section 23 for test.

A breakdown of the characteristics of the training sets of each treebank is shown in Table 1. The

[3]It should be noted that in certain cases the number of SW transitions can be reduced even further by swapping tokens that are already in the projective order. The MINIMAL oracle ensures that the number of SW transitions is minimal while still respecting the projective order.

[4]Even with 256gb of main memory we were unable to keep some of the lattices of the training sets in memory despite an efficient implementation.

[5]For the sake of reproducibility we make our implementation available on the first author's website.

	ar	de	en	eu	fr	he	hu	ko	pl	sv
# Sent.	15,762	40,472	39,832	7,577	14,759	5,000	8,146	23,010	6,578	5,000
% Proj. Sent.	97.32%	67.23%	99.90%	94.71%	99.97%	99.82%	87.75%	100%	99.54%	93.62%
# EAGER Swaps	6,481	155,041	146	984	6	12	3,654	0	91	2,062
% Swap Red. (LAZY)	80.59%	75.09%	71.92%	53.46%	16.67%	8.33%	51.07%	-	59.34%	75.90%
% Swap Red. (MINIMAL)	80.79%	83.88%	-	-	-	-	54.24%	-	-	77.79%
% Sent. w/ Unique tr. seq.	9.94%	7.81%	1.31%	1.06%	2.66%	2.82%	10.25%	0.27%	10.57%	7.28%

Table 1: Data set statistics (training sets).

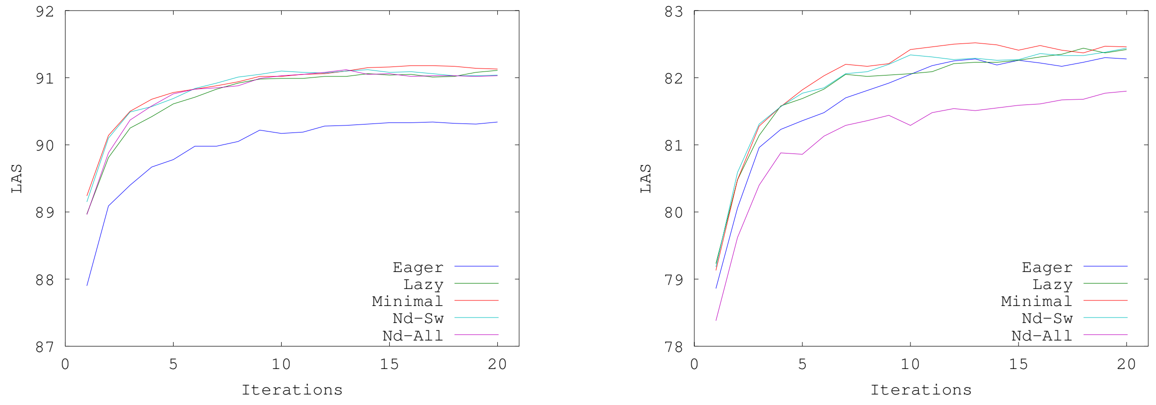


Figure 5: Learning curves of the different oracles for German (left) and Hungarian (right).

table shows the total number of sentences and the percentage of projective sentences. It also shows the total number of swap transitions required by EAGER, and the reduction of swaps of LAZY and MINIMAL relative to EAGER. For instance, in the Arabic treebank 97.32% of the sentences are projective and the LAZY and MINIMAL reduce the number of swaps by 80.59% and 80.79%, respectively. For about half the treebanks LAZY is already minimal and we exclude MINIMAL.

Korean is the only strictly projective treebank, although some of the treebanks have very few non-projective arcs in their training sets, particularly Hebrew and French. This means that the number of SH-SW ambiguities considered by the non-deterministic oracles during training is extremely small. The ND-SW oracle thus exhibits a very tiny amount of spurious ambiguity in these cases. Nevertheless, ND-ALL will still consider the SH-LA ambiguity. The last row of Table 1 shows the percentage of sentences that exhibit no spurious ambiguity under the ND-ALL oracle. This fraction ranges between almost 0% and up to about 10%, which means that there are indeed plenty of spurious ambiguities in the training data.

Preprocessing. We adopt a realistic evaluation setting and use predicted part-of-speech tags and morphological features. Specifically, we use MarMoT (Mueller et al., 2013), a state-of-the-art CRF tagger that jointly predicts part-of-speech tags and morphology. We train the parsers on 10-fold jackknifed training data. For the development and test sets the tagger is trained on the full training set.

Evaluation. We evaluate the parsers using labeled attachment score (LAS), i.e., the percentage of arcs that have the correct heads and labels. We omit the unlabeled version of this metric as we observed that it is closely correlated with LAS. We test for significance using the Wilcoxon signed rank test and mark significance at the $p < 0.05$ and $p < 0.01$ levels with † and ‡, respectively.

Training iterations. Since the parsers trained using the non-deterministic oracles rely on a latent sequence, they might require more training iterations before reaching good performance. Moreover, during initial experiments on the development data we saw that the learning curves are not monotonically increasing. To test our main hypothesis – that beam-search parsers can profit from training with a non-deterministic oracle – we tune the number of training iterations on the development sets for each oracle and treebank.

Figure 5 shows the learning curves of the beam search parser on German and Hungarian. These two plots are chosen since they paint a rather divergent picture, where EAGER is clearly underperforming for German, and ND-ALL is considerably worse than other oracles for Hungarian. For most of the other treebanks, however, the learning curves are surprisingly similar for all oracles.

	ar	de	en	eu	fr	he	hu	ko	pl	sv
ND-SW	85.92	**91.12**	**89.08**	80.53	83.49	77.89	**82.44**	-	82.50	**74.75**
ND-ALL	**86.10**	**91.12**	88.80	**80.88**	**83.66**	**78.00**	81.80	**85.18**	**83.98**	74.60
EAGER	85.88	90.34	**88.96**	80.54	83.49	77.85	82.30	**85.30**	82.74	75.01
LAZY	85.93	91.11	88.94	**80.87**	**83.65**	**77.99**	82.44	-	**82.99**	75.25
MINIMAL	**85.96**	**91.18**	-	-	-	-	**82.52**	-	-	**75.41**

Table 2: Beam search results on dev sets. The best non-deterministic and static oracles are bold.

	ar	de	en-sd	eu	fr	he	hu	ko	pl	sv
Static	85.05	87.53	90.35	79.97	83.10	78.65	83.60	85.03	82.08	79.05
Non-det.	+0.06	-0.23	+0.13	+0.55	-0.11	-0.39	+0.08	+0.09	+1.26‡	-0.07

Table 3: Test set result with beam search comparing the best static and non-deterministic oracles.

6.1 Results

Table 2 displays the LAS on the development sets for each oracle after tuning. When LAZY is already minimal, we omit MINIMAL. Since Korean is projective, we only compare ND-ALL and EAGER, which thus reduces to comparing a static and a non-deterministic oracle for ArcStandard.

The differences between the oracles are rather small. The most interesting differences occur for German, where the EAGER oracle is clearly behind, and Polish, where ND-ALL is considerably ahead of all the other oracles. Polish is also the only case where one of the non-deterministic oracles appears to be clearly ahead of the static ones.

Among the static oracles the oracle that requires the least amount of `SW` transitions generally performs best. The only exception is English, where EAGER is marginally ahead of LAZY. While the English treebank has relatively few non-projective sentences, the case is even more extreme for Hebrew and French. For these treebanks the difference between EAGER and LAZY amounts to a single swap, yet the difference in LAS is greater than 0.1%. This tiny difference in transition sequences in the training data appears to have a butterfly effect during the online learning, such that a single different update changes the outcome of the resulting weight vector to this effect.

For the non-deterministic oracles the picture is more mixed. Which oracle is better appears to be rather treebank specific, although for the most part the differences are not that big.

Finally we compare the best static with the best non-deterministic oracle on the test sets of each language. The results are shown in Table 3. In about half the cases the non-deterministic oracle does slightly better than the static one, but in the other half it is the other way around. The only significant difference is the improvement of the non-deterministic oracle for Polish. All in all, however, we conclude that static oracles generally perform as well as non-deterministic oracles.

6.2 What about greedy?

Since the non-deterministic oracles do not seem to be that helpful for the beam search parser, we wonder if the effect is the same in the greedy setting. This case has previously only been studied for projective parsers (Goldberg and Nivre, 2012; Goldberg et al., 2014). Table 4 shows the results of the greedy parser on the development sets after tuning. The greedy parser exhibits a clearer pattern compared to the beam search parser. For the non-deterministic oracles, ND-ALL is typically the best. For the static oracles the trend is that fewer swaps are better.

The final evaluation on the test sets of the greedy parser is shown in Table 5. In four cases the non-deterministic oracle is significantly better than the best static one, and overall the non-deterministic oracle is never harmful.

Comparing the results between the greedy and beam search parsers, it is clear that the greedy parser generally is behind by one or two points absolute. A peculiar exception is Korean, where the differences between the two parsers are remarkably small, yet in favor of the beam search parser.

6.3 Discussion

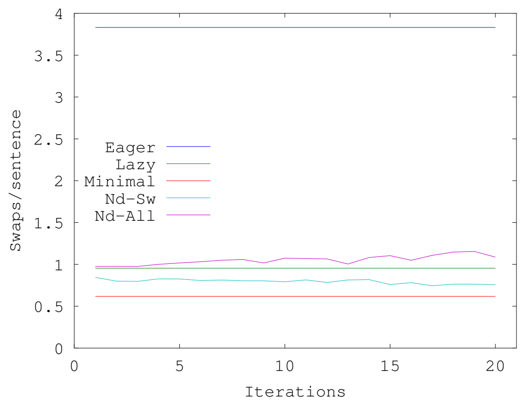
So what do the latent transition sequences look like? Figure 6 shows two plots of the average number of `SW` transitions per sentence for German and Hungarian as a function of the number of training iterations. The static oracles render straight lines since the transition sequences do not change between iterations, while the non-deterministic oracles do. Also here these two treebanks exhibit different extremes. In the case of German, the EAGER oracle is clearly applying `SW` much more

	ar	de	en	eu	fr	he	hu	ko	pl	sv
ND-SW	83.84	88.44	86.83	79.34	81.57	75.75	**79.17**	-	80.31	**72.74**
ND-ALL	**84.12**	**88.76**	**87.57**	**79.59**	**81.99**	**76.18**	78.87	**85.06**	**80.47**	72.72
EAGER	83.68	87.45	86.55	79.07	81.52	75.62	79.04	**84.92**	**80.18**	72.53
LAZY	83.74	88.45	**86.70**	**79.40**	**81.57**	**75.75**	**79.18**	-	79.60	**73.29**
MINIMAL	**83.76**	**88.81**	-	-	-	-	79.15	-	-	73.08

Table 4: Greedy results on dev sets. The best non-deterministic and static oracles are bold.

	ar	de	en	eu	fr	he	hu	ko	pl	sv
Static	82.99	84.22	87.85	78.58	81.12	75.27	81.45	84.52	79.10	75.89
Non-det.	+0.04	+0.03	+0.60[‡]	+0.24	+0.40[‡]	+0.70[†]	+0.22	+0.30	+1.33[‡]	+0.39

Table 5: Test set result with greedy search comparing the best static and non-deterministic oracles.



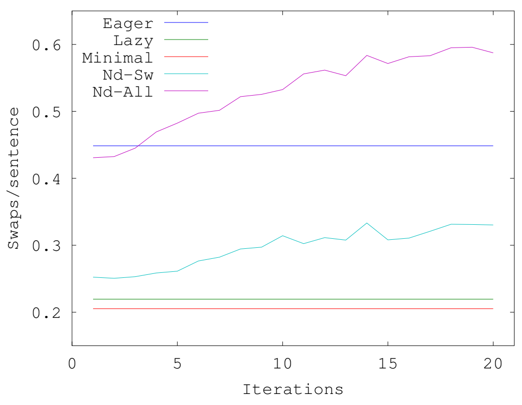


Figure 6: Average number of swaps per sentence during training for German (left) and Hungarian (right).

than any other oracle. The non-deterministic oracles tend to stay very close to the minimal number of SW transitions. For Hungarian the picture is dramatically different. The ND-ALL oracle has a tendency to overswap and gradually applies more and more SW transitions. These results bear a striking resemblance to those shown in the learning curves from Figure 5, where EAGER and ND-ALL are bad for German and Hungarian, respectively. For most of the other treebanks the corresponding curves are much closer. Indeed, as the other treebanks exhibit considerably less non-projectivity, the amount of spurious ambiguity and choice of swaps is much more constrained.

But why do the non-deterministic oracles seem to be beneficial for the greedy parser but not for the beam search parser? One reason might be that the non-deterministic oracle provides greater diversity in the training data. This is the same effect that dynamic oracles achieve with training by exploration, although it is less pronounced when only using a non-deterministic oracle. The beam search parser, on the other hand, is already exposed to many mistakes during training because of the global learning. Since the beam search parser actually does explore multiple possible transition sequences, it is probably also more lenient towards only seeing a single (static) sequence of transitions for every training instance.

7 Conclusion

The SwapStandard transition system for dependency parsing has proven very useful, especially for parsing languages with a high degree of non-projectivity, but until now it has not been known how to define non-deterministic oracles for this system. By mapping out the spurious ambiguities of the system, we have managed to solve this problem as well as the open problem of finding the minimal number of swaps using a static oracle. This has enabled us, for the first time, to evaluate the utility of non-deterministic oracles when training non-projective dependency parsers using beam search as well as greedy search. In the beam search case, the results indicate that there is no real benefit non-deterministic oracles, presumably because beam search compensates for the non-determinism. In the greedy case, we have extended previous results to the non-projective domain, showing that non-deterministic oracles are at least as good as, and sometimes significantly better than, static ones.

Acknowledgments

We are grateful to the anonymous reviewers and Agnieszka Faleńska for comments on earlier versions of this paper. We also thank Wolfgang Seeker for many valuable discussions. The first author is funded by the Deutsche Forschungsgemeinschaft (DFG) via SFB 732, project D8.

References

Giuseppe Attardi. 2006. Experiments with a multilanguage non-projective dependency parser. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 166–170, New York City, June. Association for Computational Linguistics.

Bernd Bohnet and Joakim Nivre. 2012. A transition-based system for joint part-of-speech tagging and labeled non-projective dependency parsing. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1455–1465, Jeju Island, Korea, July. Association for Computational Linguistics.

Bernd Bohnet, Joakim Nivre, Igor Boguslavsky, Richárd Farkas, Filip Ginter, and Jan Hajič. 2013. Joint morphological and syntactic analysis for richly inflected languages. *Transactions of the Association for Computational Linguistics*, 1:415–428.

Bernd Bohnet. 2010. Top accuracy and fast dependency parsing is not a contradiction. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 89–97, Beijing, China, August. Coling 2010 Organizing Committee.

Michael Collins and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 111–118, Barcelona, Spain, July.

Michael Collins. 2002. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing*, pages 1–8. Association for Computational Linguistics, July.

Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. 2006. Online passive–aggressive algorithms. *Journal of Machine Learning Reseach*, 7:551–585, March.

Marie-Catherine de Marneffe, Bill MacCartney, and Christopher D. Manning. 2006. Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC*, pages 449–454.

Eraldo Fernandes, Cícero dos Santos, and Ruy Milidiú. 2012. Latent structure perceptron with feature induction for unrestricted coreference resolution. In *Joint Conference on EMNLP and CoNLL - Shared Task*, pages 41–48, Jeju Island, Korea, July. Association for Computational Linguistics.

Yoav Goldberg and Joakim Nivre. 2012. A dynamic oracle for arc-eager dependency parsing. In *Proceedings of COLING 2012*, pages 959–976, Mumbai, India, December. The COLING 2012 Organizing Committee.

Yoav Goldberg and Joakim Nivre. 2013. Training deterministic parsers with non-deterministic oracles. *Transactions of the association for Computational Linguistics*, 1:403–414.

Yoav Goldberg, Francesco Sartorio, and Giorgio Satta. 2014. A tabular method for dynamic oracles in transition-based parsing. *Transactions of the Association for Computational Linguistics*, 2:119–130.

Carlos Gómez-Rodríguez and Joakim Nivre. 2010. A transition-based parser for 2-planar dependency structures. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1492–1501, Uppsala, Sweden, July. Association for Computational Linguistics.

Carlos Gómez-Rodríguez, Francesco Sartorio, and Giorgio Satta. 2014. A polynomial-time dynamic oracle for non-projective dependency parsing. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 917–927, Doha, Qatar, October. Association for Computational Linguistics.

Liang Huang, Suphan Fayong, and Yang Guo. 2012. Structured perceptron with inexact search. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 142–151, Montréal, Canada, 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.

Thomas Mueller, Helmut Schmid, and Hinrich Schütze. 2013. Efficient higher-order CRFs for morphological tagging. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 322–332, Seattle, Washington, USA, October. Association for Computational Linguistics.

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'09)*, pages 73–76, Paris, France, October. Association for Computational Linguistics.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies*, pages 149–160, Nancy, France.

Joakim Nivre. 2004. Incrementality in deterministic dependency parsing. In *Incremental Parsing: Bringing Engineering and Cognition Together (ACL Workshop)*, pages 50–57.

Joakim Nivre. 2009. Non-projective dependency parsing in expected linear time. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 351–359, Suntec, Singapore, August. Association for Computational Linguistics.

Francesco Sartorio, Giorgio Satta, and Joakim Nivre. 2013. A transition-based dependency parser using a dynamic parsing strategy. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 135–144, Sofia, Bulgaria, August. Association for Computational Linguistics.

Djamé Seddah, Sandra Kübler, and Reut Tsarfaty. 2014. Introducing the spmrl 2014 shared task on parsing morphologically-rich languages. In *Proceedings of the First Joint Workshop on Statistical Parsing of Morphologically Rich Languages and Syntactic Analysis of Non-Canonical Languages*, pages 103–109, Dublin, Ireland, August. Dublin City University.

Heng Yu, Liang Huang, Haitao Mi, and Kai Zhao. 2013. Max-violation perceptron and forced decoding for scalable MT training. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1112–1123, Seattle, Washington, USA, October. Association for Computational Linguistics.

Yue Zhang and Stephen Clark. 2008. A tale of two parsers: Investigating and combining graph-based and transition-based dependency parsing. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 562–571, Honolulu, Hawaii, October. Association for Computational Linguistics.

Yue Zhang and Joakim Nivre. 2011. Transition-based dependency parsing with rich non-local features. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 188–193, Portland, Oregon, USA, June. Association for Computational Linguistics.

Junsheng Zhou, Juhong Xu, and Weiguang Qu. 2013. Efficient latent structural perceptron with hybrid trees for semantic parsing. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI '13, pages 2246–2252. AAAI Press.

Enhancing the Inside-Outside Recursive Neural Network Reranker for Dependency Parsing

Phong Le
Institute for Logic, Language and Computation
University of Amsterdam, the Netherlands
p.le@uva.nl

Abstract

We propose solutions to enhance the Inside-Outside Recursive Neural Network (IORNN) reranker of Le and Zuidema (2014). Replacing the original softmax function with a hierarchical softmax using a binary tree constructed by combining output of the Brown clustering algorithm and frequency-based Huffman codes, we significantly reduce the reranker's computational complexity. In addition, enriching contexts used in the reranker by adding subtrees rooted at (ancestors') cousin nodes, the accuracy is increased.

1 Introduction

Using neural networks for syntactic parsing has become popular recently, thanks to promising results that those neural-net-based parsers achieved. For constituent parsing, Socher et al. (2013) using a recursive neural network (RNN) got an F1 score close to the state-of-the-art on the Penn WSJ corpus. For dependency parsing, the inside-outside recursive neural net (IORNN) reranker proposed by Le and Zuidema (2014) is among the top systems, including the Chen and Manning (2014)'s extremely fast transition-based parser employing a traditional feed-forward neural network.

There are many reasons why neural-net-based systems perform very well. First, Bansal et al. (2014) showed that using word-embeddings can lead to significant improvement for dependency parsing. Interestingly, those neural-net-based systems can transparently and easily employ word-embeddings by initializing their networks with those vectors. Second, by comparing a count-based model with their neural-net-based model on perplexity, Le and Zuidema (2014) suggested that predicting with neural nets is an effective solution for the problem of data sparsity. Last but not least, as showed in the work of Socher and colleagues on RNNs, e.g. Socher et al. (2013), neural networks are capable of 'semantic transfer', which is essential for disambiguation.

We focus on how to enhance the IORNN reranker of Le and Zuidema (2014) by both reducing its computational complexity and increasing its accuracy. Although this reranker is among the top systems in accuracy, its computation is very costly due to its softmax function used to compute probabilities of generating tokens: all possible words in the vocabulary are taken into account. Solutions for this are to approximate the original softmax function by using a hierarchical softmax (Morin and Bengio, 2005), noise-contrastive estimation (Mnih and Teh, 2012), or factorization using classes (Mikolov et al., 2011). A cost of using those approximations is, however, drop of the system performance. To reduce the drop, we use a hierarchical softmax with a binary tree constructed by combining Brown clusters and Huffman codes.

We show that, thanks to the reranking framework and the IORNN's ability to overcome the problem of data sparsity, more complex contexts can be employed to generate tokens. We introduce a new type of contexts, named *full-history*. By employing both the hierarchical softmax and the new type of context, our new IORNN reranker has significantly lower complexity but higher accuracy than the original reranker.

2 The IORNN Reranker

We firstly introduce the IORNN reranker (Le and Zuidema, 2014).

2.1 The ∞-order Generative Model

The reranker employs the generative model proposed by Eisner (1996). Intuitively, this model is top-down: starting with ROOT, we generate its left dependents and its right dependents. We then generate dependents for each ROOT's dependent. The generative process recursively continues until there is no dependent to generate. Formally, this model is described by the following formula

$$P(T(H)) = \prod_{l=1}^{L} P\left(H_l^L|\mathcal{C}_{H_l^L}\right) P\left(T(H_l^L)\right) \times \prod_{r=1}^{R} P\left(H_r^R|\mathcal{C}_{H_r^R}\right) P\left(T(H_r^R)\right) \quad (1)$$

where H is the current head, $T(N)$ is the subtree rooted at N, and $\mathcal{C}_N$ is the context to generate N. H^L, H^R are respectively H's left dependents and right dependents, plus EOC (End-Of-Children), a special token to inform that there are no more dependents to generate. Thus, $P(T(ROOT))$ is the probability of generating the entire T.

Le and Zuidema's ∞-order generative model is defined as the Eisner's model in which the context $\mathcal{C}_D^\infty$ to generate D contains *all* of D's generated siblings, its ancestors and theirs siblings. Because of very large fragments that contexts are allowed to hold, traditional count-based methods are impractical (even if we use smart smoothing techniques). They thus introduced the IORNN architecture to estimate the model.

2.2 Estimation with the IORNN

Each tree node y carries three vectors: inner representation $\mathbf{i}_y$, representing y, outer representation $\mathbf{o}_y$, representing the *full* context of y, and partial outer representation $\bar{\mathbf{o}}_y$, representing the *partial* context $\mathcal{C}_y^\infty$ which generates the token of y.

Without loss of generality and ignoring directions for simplicity, we assume that the model is generating dependent y for node h conditioning on context $\mathcal{C}_y^\infty$ (see Figure 1). Under the approximation that the inner representation of a phrase equals the inner representation of its head, and thanks to the recursive definition of full/partial contexts ($\mathcal{C}_y^\infty$ is a combination of $\mathcal{C}_h^\infty$ and y's previously generated sisters), the (partial) outer representations of y are computed as follows.

$$\bar{\mathbf{o}}_y = f(\mathbf{W}_{hi}\mathbf{i}_h + \mathbf{W}_{ho}\mathbf{o}_h + \mathbf{b}_o + \mathbf{r})$$

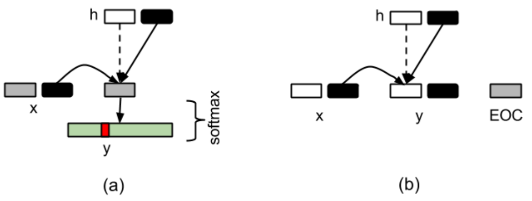


Figure 1: The process to (a) generate y, (b) compute outer representation $\mathbf{o}_y$, given head h and sibling x. Black, grey, white boxes are respectively inner, partial outer, and outer representations. (Le and Zuidema, 2014)

where $\mathbf{r} = \mathbf{0}$ if y is the first dependent of h; otherwise, $\mathbf{r} = \frac{1}{|\bar{\mathcal{S}}(y)|}\sum_{v\in\bar{\mathcal{S}}(y)} \mathbf{W}_{dr(v)}\mathbf{i}_v$, where $\bar{\mathcal{S}}(y)$ is the set of y's sisters generated before. And,

$$\mathbf{o}_y = f(\mathbf{W}_{hi}\mathbf{i}_h + \mathbf{W}_{ho}\mathbf{o}_h + \mathbf{b}_o + \mathbf{s})$$

where $\mathbf{s} = \mathbf{0}$ if y is the only dependent of h (ignoring EOC); otherwise $\mathbf{s} = \frac{1}{|\mathcal{S}(y)|}\sum_{v\in\mathcal{S}(y)} \mathbf{W}_{dr(v)}\mathbf{i}_v$ where $\mathcal{S}(y)$ is the set of y's sisters. $dr(v)$ is the dependency relation of v with h. $\mathbf{W}_{hi/ho/dr(v)}$ are $n \times n$ real matrices, and $\mathbf{b}_o$ is an n-d real vector.

The probability $P(w|\mathcal{C}_y^\infty)$ of generating a token w at node y is given by

$$softmax(w, \bar{\mathbf{o}}_y) = \frac{e^{u(w,\bar{\mathbf{o}}_y)}}{\sum_{w'\in V} e^{u(w',\bar{\mathbf{o}}_y)}} \quad (2)$$

where $\left[u(w_1, \bar{\mathbf{o}}_y), ..., u(w_{|V|}, \bar{\mathbf{o}}_y)\right]^T = \mathbf{W}\bar{\mathbf{o}}_y + \mathbf{b}$ and V is the set of all possible tokens (i.e. vocabulary). $\mathbf{W}$ is a $|V| \times n$ real matrix, $\mathbf{b}$ is an $|V|$-d real vector.

2.3 The Reranker

Le and Zuidema's (mixture) reranker is

$$T^* = \arg\max_{T\in\mathcal{D}(S)} \alpha \log P(T(ROOT)) + (1-\alpha)s(S,T) \quad (3)$$

where $\mathcal{D}(S)$ and $s(S,T)$ are a k-best list and scores given by a third-party parser, and $\alpha \in [0,1]$.

3 Reduce Complexity

The complexity of the IORNN reranker above for computing $P(T(ROOT))$ is approximately[1]

$$O = l \times (3 \times n \times n + n \times |V|)$$

[1]Look at Figure 1, we can see that each node requires four matrix-vector multiplications: two for computing children's (partial) outer representation, one for computing sisters' (partial) outer representations, and one for computing the softmax.

where l is the length of the given sentence, n and $|V|$ are respectively the dimensions of representations and the vocabulary size (sums of vectors are ignored because their computational cost is small w.r.t $l \times n \times n$). In Le and Zuidema's reranker, $|V| \approx 14000 \gg n = 200$. It means that the reranker spends most of its time on computing $softmax(w, \bar{\mathbf{o}}_y)$ in Equation 2. This is also true for the complexity in the training phase.

To reduce the reranker's complexity, we need to approximate this softmax. Mnih and Teh (2012) propose using the noise-contrastive estimation method which is to force the system to discriminate correct words from randomly chosen candidates (i.e., noise). This approach is very fast in training thanks to fixing normalization factors to one, but slow in testing because normalization factors are explicitly computed. Vaswani et al. (2013) use the same approach, and also fix normalization factors to one when testing. This, however, doest not guarantee to give us properly normalized probabilities. We thus employ the hierarchical sofmax proposed by Morin and Bengio (2005) which is fast in both training and testing and outputs properly normalized probabilities.

Assume that there is a binary tree whose leaf nodes each correspond to a word in the vocabulary. Let $(u_1^w, u_2^w, ..., u_L^w)$ be a path from the root to the leaf node w (i.e. $u_1^w =$ root and $u_L^w = w$). Let $L(u)$ the left child of node u, and $[x]$ be 1 if x true and -1 otherwise. We then replace Equation 2 by

$$P(w|\mathcal{C}_y^{\infty}) = \prod_{i=1}^{L-1} \sigma\big([u_{i+1}^w = L(u_i^w)]\mathbf{v}_{u_i^w}^T \times \bar{\mathbf{o}}_y\big)$$

where $\sigma(z) = 1/(1+e^{-z})$. If the binary tree is perfectly balanced, the new complexity is approximately $l \times (3 \times n \times n + n \times \log(|V|))$, which is less than $4l \times n \times n$ if $|V| < 2^n$ (1.6×10^{60} as $n = 200$ in the Le and Zuidema's reranker).

Constructing a binary tree for this hierarchical softmax turns out to be nontrivial. Morin and Bengio (2005) relied on WordNet whereas Mikolov et al. (2013) used only frequency-based Huffmann codes. In our case, an ideal tree should reflect both semantic similarities between words (e.g. leaf nodes for 'dog' and 'cat' should be close to each other), and word frequencies (since we want to minimize the complexity). Therefore we propose combining output of the Brown hierarchical clustering algorithm (Brown et al., 1992) and frequency-based Huffman codes.[2] Firstly, we use the Brown algorithm to find c hierarchical clusters ($c = 500$ in our experiments).[3] We then, for each cluster, compute the Huffman code for each word in that cluster.

4 Enrich Contexts

Although suggesting that predicting with neural networks is a solution to overcome the problem of sparsity, Le and Zuidema's reranker still relies on two widely-used independence assumptions: (i) the two Markov chains that generate dependents in the two directions are independent, given the head, and (ii) non-overlapping subtrees are generated independently.[4] That is why its partial context (e.g. the red-dashed shape in Figure 2) used to generate a node ignores: (i) sisters in the other direction and (ii) ancestors' cousins and their descendants.

We, in contrast, eliminate those two assumptions by proposing the following top-down left-to-right generative story. From the head node, we generate its dependents from left to right. The partial context to generate a dependent is the whole fragment that is generated so far (e.g. the blue shape in Figure 2). We then generate subtrees rooted at those nodes also from left to right. The full context given to a node to generate the subtree rooted at this node is thus the whole fragment that is generated so far (e.g. the combination of the blue shape and the blue-dotted shape in Figure 2). In this way, the model always uses the whole up-to-date fragment to generate a dependent or to generate a subtree rooted at a node. To our knowledge, these contexts, which contain full derivation histories, are the most complete ones ever used for graph-based parsing.

Extending the IORNN reranker in this way is straight-forward. For example, we first generate a subtree $tr(x)$ rooted at node x in Figure 3. We then compute the inner representation for $tr(x)$: if $tr(x)$ contains only x then $\mathbf{i}_{tr(x)} = \mathbf{i}_x$; otherwise

$$\mathbf{i}_{tr(x)} = f(\mathbf{W}_h^i \mathbf{i}_x + \frac{1}{|\mathcal{S}(x)|} \sum_{u \in \mathcal{S}(x)} \mathbf{W}_{dr(u)}^i \mathbf{i}_{tr(u)} + \mathbf{b}^i)$$

[2]Another reason not to use the Brown clustering algorithm alone is that the algorithm is inefficient with high numbers of clusters ($|V| \approx 14000$ in this case).

[3]We run Liang (2005)'s implementation at `https://github.com/percyliang/brown-cluster` on the training data.

[4]Le and Zuidema rephrased this assumption by the approximation that the meaning of a phrase equals to the meaning of its head.

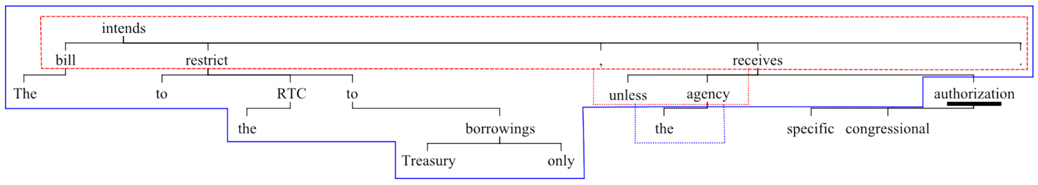


Figure 2: Context used in Le and Zuidema's reranker (red-dashed shape) and full-history context (blue-solid shape) to generate token 'authorization'.

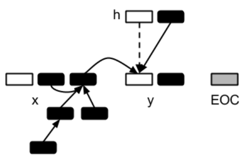


Figure 3: Compute the full-history outer representation for y.

where $S(x)$ is the set of x's dependents, $dr(u)$ is the dependency relation of u with x, $\mathbf{W}^i_{h/dr(u)}$ are $n \times n$ real matrices, and $\mathbf{b}^i$ is an n-d real vector.[5]

5 Experiments

We use the same setting reported in Le and Zuidema (2014, Section 5.3). The Penn WSJ Treebank is converted to dependencies using the Yamada and Matsumoto (2003)'s head rules. Sections 2-21 are for training, section 22 for development, and section 23 for testing. The development and test sets are tagged by the Stanford POS-tagger trained on the whole training data, whereas 10-way jackknifing is used to generate tags for the training set. For the new IORNN reranker, we set $n = 200$, initialise it with the 50-dim word embeddings from Collobert et al. (2011). We use the MSTParser (with the 2nd-order feature mode) (McDonald et al., 2005) to generate k-best lists, and optimize k and α (Equation 3) on the development set.

Table 1 shows the comparison of our new reranker against other systems. It is a surprise that our reranker with the proposed hierarchical softmax alone can achieve an almost equivalent score with Le and Zuidema's reranker. We conjecture that drawbacks of the hierarchical softmax compared to the original can be lessened by probabilities of generating other elements like POS-tags, dependency relations. Adding enriched contexts, our reranker achieves the second best accuracy among those systems.

[5]Note that the overall computational complexity increases linearly with $l \times n \times n$. For instance, for computing $P(T(ROOT))$, the increase is approximately $2l \times n \times n$ since each node requires maximally two more matrix-vector multiplications.

System	UAS
MSTParser (baseline)	92.06
Koo and Collins (2010)	93.04
Zhang and McDonald (2012)	93.06
Martins et al. (2013)	93.07
Bohnet and Kuhn (2012)	93.39
Reranking	
Hayashi et al. (2013)	93.12
Le and Zuidema (2014)	93.12
Our reranker (h-softmax only, $k = 45$)	93.10
Our reranker ($k = 47$)	**93.27**

Table 1: Comparison with other systems on section 23 (excluding punctuation).

Because in this experiment no words have paths longer than $20 \ll n = 200$, our new reranker has a significantly lower complexity than the one of Le and Zuidema's reranker. On a computer with an Intel Core-i5 3.3GHz CPU and 8GB RAM, it takes 20 minutes to train this reranker, which is implemented in C++, and 2 minutes to evaluate it on the test set.

6 Conclusion

Solutions to enhance the IORNN reranker of Le and Zuidema (2014) were proposed. We showed that, by replacing the original softmax function with a hierarchical softmax, the reranker's computational complexity significantly decreases. The cost of this, which is drop on accuracy, is avoided by enriching contexts with subtrees rooted at (ancestors') cousin nodes. The new reranker, according to experimental results on the Penn WSJ Treebank, has even higher accuracy than the old one.

Acknowledgments

We thank Willem Zuidema and four anonymous reviewers for helpful comments.

References

Mohit Bansal, Kevin Gimpel, and Karen Livescu. 2014. Tailoring continuous word representations for dependency parsing. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Bernd Bohnet and Jonas Kuhn. 2012. The best of both worlds: a graph-based completion model for transition-based parsers. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 77–87. Association for Computational Linguistics.

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.

Danqi Chen and Christopher D Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 740–750.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, 12:2493–2537.

Jason M Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th conference on Computational linguistics-Volume 1*, pages 340–345. Association for Computational Linguistics.

Katsuhiko Hayashi, Shuhei Kondo, and Yuji Matsumoto. 2013. Efficient stacked dependency parsing by forest reranking. *Transactions of the Association for Computational Linguistics*, 1(1):139–150.

Terry Koo and Michael Collins. 2010. Efficient third-order dependency parsers. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1–11. Association for Computational Linguistics.

Phong Le and Willem Zuidema. 2014. The inside-outside recursive neural network model for dependency parsing. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

Percy Liang. 2005. Semi-supervised learning for natural language. In *MASTERS THESIS, MIT.*

Andre Martins, Miguel Almeida, and Noah A. Smith. 2013. Turning on the turbo: Fast third-order non-projective turbo parsers. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 617–622, Sofia, Bulgaria, August. Association for Computational Linguistics.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005. Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 91–98. Association for Computational Linguistics.

Tomas Mikolov, Stefan Kombrink, Lukas Burget, JH Cernocky, and Sanjeev Khudanpur. 2011. Extensions of recurrent neural network language model. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 5528–5531. IEEE.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119.

Andriy Mnih and Yee Whye Teh. 2012. A fast and simple algorithm for training neural probabilistic language models. In *Proceedings of the 29th International Conference on Machine Learning*, pages 1751–1758.

Frederic Morin and Yoshua Bengio. 2005. Hierarchical probabilistic neural network language model. In *AISTATS*, volume 5, pages 246–252. Citeseer.

Richard Socher, John Bauer, Christopher D Manning, and Andrew Y Ng. 2013. Parsing with compositional vector grammars. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics*, pages 455–465.

Ashish Vaswani, Yinggong Zhao, Victoria Fossum, and David Chiang. 2013. Decoding with large-scale neural language models improves translation. In *EMNLP*, pages 1387–1392. Citeseer.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistical dependency analysis with support vector machines. In *Proceedings of International Conference on Parsing Technologies (IWPT)*, pages 195–206.

Hao Zhang and Ryan McDonald. 2012. Generalized higher-order dependency parsing with cube pruning. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 320–331. Association for Computational Linguistics.

Maximising spanning subtree scores for parsing tree approximations of semantic dependency digraphs

Natalie Schluter
Center for Language Technology
University of Copenhagen
natschluter@hum.ku.dk

Abstract

We present a method for finding the best tree approximation parse of a dependency digraph for a given sentence, with respect to a dataset of semantic digraphs as a computationally efficient and accurate alternative to DAG parsing. We present a training algorithm that learns the spanning subtree parses with the highest scores with respect to the data, and consider the output of this algorithm a description of the best tree approximations for digraphs of sentences from similar data. With the results from this approach, we acquire some important insights on the limits of solely data-driven tree approximation approaches to semantic dependency DAG parsing, and their rule-based, pre-processed tree approximation counterparts.

1 Introduction

In semantic dependency parsing, the aim is to recover sentence-internal predicate argument relationships; structurally speaking, given a sentence, the objective is to recover the possibly disconnected digraph (which represents the semantic structure of the sentence). The sparsity of digraph representations of some semantic dependency digraph datasets (i.e., the fact that the number of edges is linear in the number of vertices), as well as the well-performing first attempts at such tree approximations from Schluter et al. (2014), Agić and Koller (2014), and Agić et al. (2015) suggest that tree approximations for digraph semantic dependency structures are a relevant avenue to the development if not the sidestepping of some computationally harder models of directed acyclic graph (DAG) and digraph decoding (McDonald and Pereira, 2006; Martins and Almeida, 2014).

In this paper, we present a simple adaptation of the passive-aggressive perceptron training algorithm used in (Crammer et al., 2006; Björkelund et al., 2009) to the task of finding the parameters that describe the highest scoring tree approximations of semantic dependency digraphs, given a training corpus. The key change in the algorithm is to iteratively minimise the error in precision between output spanning subtrees and corresponding training digraph instances, allowing therefore the algorithm to choose best spanning subtree approximations with respect to the dataset, rather than forming tree approximations as a pre-processing step to training as was done by Schluter et al. (2014), Agić and Koller (2014), and Agić et al. (2015).

Because we directly adapt the software used by (Björkelund et al., 2009), without increasing any computational complexity, the approach also benefits from a syntactic parsing algorithm optimised both practically and theoretically for efficiency, robustness, and accuracy. Supposing these natural tree patterns exist in the data, the approach intuitively is very attractive, since it lets natural patterns of the data dictate tree approximations, rather than depending on either the previous knowledge of parser behaviour or anecdotal linguistic knowledge. Moreover, the approach promises some insight into the nature of the patterns inherent in the semantic digraph data, reflecting, namely, the question of the existence of unique most likely sub-tree structures in the data and why rule-based pre-processing tree approximations work so well for semantic digraph dependency parsing.

2 Previous work and motivation

In the context of the SemEval task 8 on Broad Coverage Semantic Dependency Parsing, Schluter et al. (2014) and Agić and Koller (2014) introduce a tree approximation definition for the semantic dependency parsing task, including a number of different approaches. Both present similar pruning pre-processing steps as a potential approach. Additionally, Schluter et al. (2014) present a packing pre-processing step as a further approach. We discuss the approaches presented in (Schluter et al., 2014), because we adopt the parts of their pre-processing which do not remove any edges. Their approaches also outperformed that of (Agić and Koller, 2014) in the SemEval task.

In (Schluter et al., 2014), all digraphs were first transformed into rooted DAGs (that is, with a single root, from which all nodes are reachable), following which the authors experimented with creating the tree approximations for each of the training set digraphs individually as follows.

In their first approach, (Schluter et al., 2014) consider underlying undirected cycles of digraphs and **pruned** these digraphs by removing from the cycle the longest edge from the node with the fewest predecessors (lowest depth) in the digraph. In their second approach, they attempt to store almost all digraph information into the graph, by **packing** parallel path information, structurally corresponding to long-distance dependency re-entrancies, into a single complex edge label; these removed edges can then recovered as a post-processing step, by "expanding" the complex edge label. However, not all parallel path sets between two nodes include a path of length 1 edge; for these, an heuristic approach was taken whereby only the edge of shortest span from among all of the last edges of these paths was retained.

The result from both these approaches were trees that were used as tree approximations to training set digraphs, individually. The packing approach suffered from complex edge labels with low frequency in the training set, and which could not always be resolved fully in post-processing at test-time; as a result, this approach had relatively stable precision and recall (with respect to each other), but with lower precision than the pruning approach, which achieved high precision, but low recall. Upon carrying out this research, we posited that the lower recall of the pruning approach was caused by not accounting for the tree approximations with respect to the entire dataset, rather only with respect to a structural heuristic on individual digraphs of the training set.

The motivation for this work is therefore an attempt to attain similar precision to the pruning approach in (Schluter et al., 2014), with similar recall to the same authors' packing approach, by only pruning edges, but leaving it to the parser to determine, given the training data, which edges should be pruned. The sparsity of the graphs in all three datasets (Cf. Section 4) as well as the disparity in precision and recall between the approaches in (Schluter et al., 2014) (Cf. Section 5) suggests that there is room for improvement in tree approximation approaches to semantic dependency parsing for these particular datasets.

Further improvements to the pre-processing approach to tree approximation parsing of semantic digraphs has been obtained by Agić et al. (2015). This approach was guided by both the ideas of pruning and packing: they first present a study on the types of re-entrancies displayed by the digraph data, to discover that many of them are predictable. Such re-entrancy information needs not be packed into other labels as it is done by (Schluter et al., 2014), as this pruning is nearly 100% reversible. These observations provide further insight into the performance of the approach presented here, given its behaviour as well as the dataset used. Indeed, persistent non-tree-like structures in the training data will inhibit training algorithms from aggressively deciding on a best tree-like substructure (Cf. Section 5).

The (non-ensemble approach) state of the art on the datasets we experiment on is achieved using a second-order model with approximate digraph decoding via alternate directions dual decomposition (Martins and Almeida, 2014; Martins et al., 2011).

3 Finding the best maximum spanning subtree approximation

As an approximate approach to finding the best scoring semantic dependency parse digraph given a training corpus of such digraphs, we present an approach to finding the best scoring semantic dependency spanning subtree parse, given a training corpus of semantic dependency digraphs. That is, we adopt a second-order approach to the problem of finding the best semantic dependency parse digraph. Our original objective was to find

$$D^* = \arg \max_{D \in \mathcal{D}(x)} \phi_D(\mathbf{w}, x, D),$$

where $\mathbf{w}$ is a weight vector, D is a dependency digraph and x is the input sentence. The best dependency digraph is the output according to a scoring function ϕ_D—a sum over sub-factors representing some second-order description of the dependency structure. The parameter $\mathbf{w}$ is obtained by some algorithm that minimises the error defined by a distance function between digraph parse D'_i and training digraph D_i.

Our aim here is to approximate D^* by a tree T^*, so our objective now concerns possible trees $T \in \mathcal{T}$; we want to find the tree that maximises the scoring function ϕ_T for trees,

$$T^* = \arg \max_{T \in \mathcal{T}(x)} \phi_T(\mathbf{w}, x, T),$$

where ϕ_T is composed of second-order factors describing (and, practically speaking, coincides precisely with the scoring function ϕ from (Carreras, 2007)). However, we obtain the parameter $\mathbf{w}$ by training on a dataset of dependency digraphs and carefully minimising the error between these two very different types of structures, since a tree can never contain more than $|x| - 1$ edges and there is therefore a risk of non-convergence if the wrong error measure is chosen.

We directly adapt the mate parser (Bohnet, 2010) for this modified task. To do this, we simply adjust the unregularised passive-aggressive perceptron algorithm implementation used in training by the parser, the averaged-perceptron version of which was presented first in (Carreras, 2007), in three key ways. First, edge features are taken from entire digraphs, rather than just trees. These features, unlike in (Martins and Almeida, 2014), do not account for multiple heads in digraphs, because we need to use the scoring function on trees rather than digraphs. Secondly, the error is minimised between the original graphs and the maximum spanning subtree implied by $\mathbf{w}$. In doing so, the algorithm finds weight vectors that minimise the error of the tree approximation of the graph. Finally, we changed the error function from (1−recall) in the original version of the mate parser to (1−precision) to avoid punishing trees for not being digraphs in the updates, and thereby prevent non-convergence.

Algorithm 1 shows the adaptation, where an instance is a pair (x_j, D_j) of x_j, a sentence, and D_j, a dependency digraph. The algorithm begins by initialising the weight vector $\mathbf{w}$ to the zero vector. It then extracts the features from the training set, $Z : \{(x_j, D_j)\}_{j=1}^{N}$ and stores them on the hard disk, after which training is carried out using the passive-aggressive algorithm. Iteratively, it (1) reads in the features and calculates possible edge weights, (2) decodes using the second order extension of the Eisner algorithm for projective dependency trees (Eisner, 1996) presented in (Carreras, 2007), and then (3) tries to find a higher scoring non-projective tree by exchanging edges out of the output from (2) using the edge weights in (1). Details for the original algorithm can be found (Bohnet, 2010); however, note that in Algorithm 1, $\hat{T}$ is a dependency tree and D_j is a dependency digraph. Also, error is measured in terms of precision (for Lines 13 and 14).

Algorithm 1 `Training` (Z)
// where $Z = \{(x_j, y_j)\}_{j=1}^{N}$ is the digraph training data

```
1: $\mathbf{w} \leftarrow \mathbf{0}$
2: for $j \leftarrow 1$ to $N$ do
3:     extract-and-store-features($x_j$)
4: end for
5: for $i \leftarrow 1$ to $I$ do
6:         // where $I$ is the number of iterations
7:     for $j \leftarrow 1$ to $N$ do
8:         $k \leftarrow (i-1) * N + j$
9:         $\gamma \leftarrow I \times N - k + 2$    // passive-aggressive weight
10:        $A \leftarrow$ read-features-and-calc-arrays($j$, $\mathbf{w}$)
11:        $\hat{T} \leftarrow$ predict-projective-parse-tree($A$)
12:        $\hat{T} \leftarrow$ non-projective-approx($\hat{T}$, $A$)
13:        $e \leftarrow \Delta(\hat{T}, D_j)$   // the error
14:        $\mathbf{w} \leftarrow$ update($e$, $\gamma$)
15:    end for
16: end for
```

4 The graphs and their tree-likeness

Organisers for the SemEval task 8 on Broad Coverage Semantic Dependency Parsing (SDP) (Oepen et al., 2014) proposed three different annotations for evaluation of the task, resulting in the three semantically annotated datasets, over the same text—that is, the Wall Street Journal section of the English Penn Treebank. As in the original task, we refer to the datasets as

DM: a transformation of Flickinger et al. (2012)'s DeepBank by Miyao et al. (2014)'s system.

PAS: the predicate-argument structures of the WSJ portion of the HPSG treebank.

PCEDT: the tectogrammatical layer of annotations from the Prague Czech-English Dependency Bank (Hajič et al., 2012).

Graph sparsity and treewidth. We note that the average number of edges of a spanning subtree for training set graphs is 22.93. On the other hand, after the aforementioned pre-processing, the average number of edges in the DM-annotated treebank is 23.77, PAS-annotated treebank is 24.32, and for PCEDT it is 23.33. By removing edges from digraphs to make spanning subtrees, we thus lose at most 5.7% edges, showing the general sparsity of digraphs in these datasets and further showing the relevance of the tree approximation approach. Agić et al. (2015) calculate the average treewidth of the underlying undirected graphs of SDP digraphs, to be 1.3 for DM, 1.71 for PAS, and 1.45, for PCEDT, indicating that DAG parsing algorithms heavily based on the more efficient tree parsing algorithms are a promising avenue for further research.

4.1 Pre-processing the graph data

In general, the digraphs amongst three datasets are disconnected. In order for the solution of finding a most likely tree approximation to make any sense, therefore, we transformed digraphs similarly to the approach taken in (Schluter et al., 2014): (1) a dummy root node is placed to the left of the input, (2) the top node is connected (as a child) to the dummy root node, (3) the node of highest degree (=indegree + outdegree) for non-singleton remaining weakly connected components is attached as a child of the dummy root node, and (4) all singleton weakly connected components are connected a child of the node to the left.

In most digraphs of of the three datasets, there is not any existing spanning subtree. Therefore, we carry out flow reversal for rooted DAG construction as a further pre-processing step. As in (Schluter et al., 2014), we created rooted DAGs during a breadth-first search of the digraph, reversing the direction of edges when necessary, and marking the label of reversed edges (for reversibility of the transformation). So, our label sets for the three datasets may at most double, which admittedly increases (by a factor of 2) instance size and therefore running time.

5 Experiments and error analysis

For our experiments, we used precisely the same data split as in SemEval 2014's task 8 and the original mate parser default parameters for the modified version, with the exception that we increased the number of iterations to 15. The results are given in Table 1. Compared with the preprocessing tree approximations from (Schluter et al., 2014), the subgraph score maximisation approach performs quite poorly. The approach successfully closes the gap between precision and recall, when compared to the pruning approach in (Schluter et al., 2014), but both precision and recall are relatively low.

	data	LP	LR	LF	UP	UR	UF
pack	DM	84.8	84.0	84.4	86.8	86.0	86.4
	PAS	87.7	88.4	88.0	89.1	89.8	89.4
	PCEDT	71.2	68.6	69.9	84.8	81.8	83.2
prune	DM	87.2	80.2	83.6	89.2	82.0	85.4
	PAS	91.3	81.3	86.0	92.6	82.5	87.3
	PCEDT	72.8	62.8	67.4	88.2	76.1	81.7
this paper	DM	67.2	69.7	68.4	69.8	72.4	71.1
	PAS	83.1	77.5	80.2	86.4	80.7	83.6
	PCEDT	62.3	58.5	60.4	79.0	74.3	76.6

Table 1: Precision, recall and f-score over the three datasets (pack and prune results are from (Schluter et al., 2014)) .

An analysis of the pre-processing (Schluter et al., 2014; Agić and Koller, 2014; Agić et al., 2015) versus statistical (this paper) approaches may provide some insight as to why. Especially Agić et al. (2015) show that many non-tree-like structures of the SDP data are predictable; that is, we can turn them into trees by consistently pruning edges and "understand" that they are in fact more complex than trees, without encoding this information into the resulting pruned tree in any way. In a post-processing step, these edges are simply reintroduced using some rules. These are edges for which it would be difficult for the passive aggressive algorithm that we employ to choose between, since there are virtually no structures that require only a subset of them. As a result, making a strict rule about what tree structures should be predicted as a pre-processing step results in better tree approximations than the purely data-driven approach presented here.

6 Concluding remarks

We have presented an approach to semantic dependency parsing that takes advantage of and subtly adapts an efficient and highly optimised syntactic dependency tree parsing system to the job of finding best tree approximations of digraphs. Intuitively, the approach is attractive, because it requires the data to choose digraph approximations, rather than using any anecdotal linguistic knowledge to hard-code a pre-processor to build the approximations before training. However, the approach fails to outperform rule-based preprocessing for tree approximations, seemingly because there is often no clear statistical preference among various subtrees of DAGs in the SDP data. The performance of a combination of the data-driven and rule-based preprocessing methods remains a viable open question.

References

Željko Agić and Alexander Koller. 2014. Potsdam: Semantic dependency parsing by bidirectional graph-tree transformations and syntactic parsing. In *Proc of Semeval*, Dublin, Ireland.

Željko Agić, Alexander Koller, and Stephan Oepen. 2015. Semantic dependency graph parsing using tree approximations. In *Proc of IWCS*, London, UK.

Anders Björkelund, Love Hafdell, and Pierre Nugues. 2009. Multilingual semantic role labeling. In *CoNLL*.

Bernd Bohnet. 2010. Top accuracy and fast dependency parsing is not a contradiction. In *Proceedings of COLING*, pages 89–97.

Xavier Carreras. 2007. Experiments with a higher-order projective dependency parser. In *Proc. of CoNLL Shared Task Session of EMNLP-CoNLL*, pages 957–961, Prague, Czech Republic.

Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. 2006. Online passive-aggressive algorithms. *JMLR*, 7:551–585.

Jason Eisner. 1996. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of COLING-96*, pages 340–345, Copenhagen, Denmark.

Daniel Flickinger, Yi Zhang, and Valia Kordoni. 2012. Deep-bank. a dynamically annotated treebank of the wall street journal. In *Proc. of TLT*, pages 85–96, Lisbon, Portugal.

Jan Hajič, Eva Hajičová, Jarmila Panevová, Petr Sgall, Ondřej Bojar, Silvie Cinková, ..., and Zdeněk Žaborkrtský. 2012. Announcing prague czech-english dependency treebank 2.0. In *Proc. of LREC*, pages 3153–3160, Istanbul, Turkey.

André F. T. Martins and Mariana S. C. Almeida. 2014. Priberam: A turbo semantic parser with second order features. In *Proc of SemEval*, Dublin, Ireland.

André Martins, Noah A. Smith, M. A. T. Figueiredo, and P. M. Q. Aguiar. 2011. Dual decomposition with many overlapping components. In *Proceedings of EMNLP*.

Ryan McDonald and Fernando Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *Proc. of EACL*, pages 81–88.

Yusuke Miyao, Stephan Oepen, and Daniel Zeman. 2014. In-house: An ensemble of pre-existing off-the-shelf parsers. In *Proc. of SemEval*.

Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Dan Flickinger, Jan Hajič, Angelina Ivanova, and Yi Zhang. 2014. Semeval 2014 task 8: Broad-coverage semantic dependency parsing. In *Proc. of SemEval*, pages 63–71, Dublin, Ireland.

Natalie Schluter, Anders Søgaard, Jakob Elming, Dirk Hovy, Barbara Plank, Hector Martinez Alonso, Anders Johanssen, and Sigrid Klerke. 2014. Copenhagen: Tree approximations of semantic parsing problems. In *Proceedings of SemEval*, Dublin, Ireland.

CKY Parsing With Independence Constraints

Joseph Irwin
Graduate School of Information Science
Nara Institute of Science and Technology
Nara, Japan
joseph-i@is.naist.jp

Yuji Matsumoto
Graduate School of Information Science
Nara Institute of Science and Technology
Nara, Japan
matsu@is.naist.jp

Abstract

The CKY algorithm is an important component in many natural language parsers. We propose a novel type of constraint for context-free parsing called independence constraints. Based on the concept of independence between words, we show how these constraints can be used to reduce the work done in the CKY algorithm. We demonstrate a classifier which can be used to identify boundaries between independent words in a sentence using only surface features, and show that it can be used to speed up a CKY parser. We investigate the trade-off between speed and accuracy, and indicate directions for improvement.

1 Introduction

The CKY algorithm is an $\mathrm{O}\big(|G|n^3\big)$ dynamic programming algorithm for finding all of the possible derivations of a sentence in a context-free language. Its complexity depends on both the sentence length n and the size of the grammar $|G|$. Methods for improving parsing accuracy typically increase the size of the grammar (Klein and Manning, 2003; Petrov and Klein, 2007) or even the exponent of n (Eisner and Satta, 1999). More powerful "deep" grammar formalisms multiply the computational complexity even more (Bangalore and Joshi, 1999).

A common technique for speeding up such parsers is coarse-to-fine parsing, where input is first parsed using a much simpler (and thus smaller) grammar, and the content of the chart is then used to constrain the search over the final grammar (Torisawa et al., 2000; Charniak and Johnson, 2005; Petrov and Klein, 2007). Even with a much smaller grammar, the CKY algorithm may be expensive—Roark et al. (2012) report that the initial CKY step in the Berkeley Parser takes half of the total parse time.

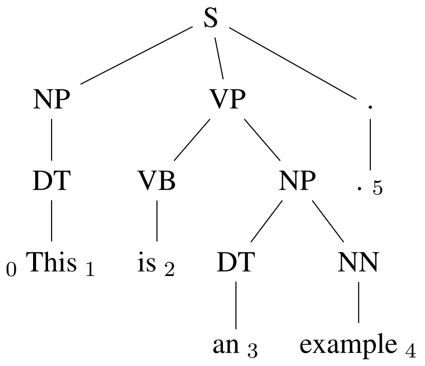


Figure 1: In this tree 'This' and 'is' are independent, while 'is' and 'an' are not.

Other techniques can be used to prune cells in the chart. Roark et al. (2012) use a finite-state model to label words that do/don't begin/end spans, and skip cells that don't satisfy the labels. Bodenstab et al. (2011) directly apply a classifier to each cell to decide how many spans to keep. Both approaches reduce the work done by the parser while preserving accuracy.

We propose a novel type of top-down constraint for a CFG parser that we call independence constraints, described in Section 2. In Section 3 we show how the CKY algorithm can be easily modified to accommodate these constraints, and in Section 4 we describe a classifier which can provide the constraints to a parser. We integrate the constraints into the Stanford Parser CKY implementation and show the results in Section 5.

2 Independence Constraints

We propose a concept we call **independence**. Given a sentence $s = w_1 w_2 \dots w_n$ and a context-free derivation (parse tree) t of s, words w_i and w_{i+1} are **independent** if every node in t that dominates both w_i and w_{i+1} also dominates w_1 and w_n. Furthermore, if w_i and w_{i+1} are independent, then $\forall j, k$ s.t. $j \leq i$ and $k > i$, w_j and w_k are independent. Less formally, if the children of the top node of a parse tree were split into separate subtrees, two words are independent if they would end up in different subtrees.

An example is shown in Figure 1. Here, 'This' and 'is' are independent, as are 'example' and '.'. The independent spans are ('This'), ('is', 'an', 'example'), and ('.'), with boundaries 1 and 4. The independent spans and independent span boundaries can be derived straightforwardly from the definition of independent words: the locations between consecutive words which are independent are the independent span boundaries, and the independent spans are simply the spans in between consecutive boundaries.

3 Modifying The CKY Algorithm With Independence Constraints

Conceptually, if a CKY parser knows the locations of the independent span boundaries for a sentence, it can perform the normal CKY algorithm for each independent span separately, and simply join the spans at the top of the tree to finish the parse, thereby avoiding work which would otherwise be done while still obtaining the desired 1-best parse. Two issues make the task more complicated than this.

The first complication is that if we assume that the independent boundaries will be identified automatically, we must allow for errors. If a location which is not an independent span boundary is given as one, the parser will make an error it would not have otherwise. On the other hand, if a location which is an independent span boundary is not marked as such, the parser may account for this at the cost of not achieving the minimum computation possible. By allowing for this second type of error, the algorithm is made more robust, and allows the independent boundary identification step to prioritize precision over recall to lessen negative impact on the parser's accuracy.

The second issue is caused by the binarization of the context-free grammar used in the CKY algorithm. Because the CKY algorithm requires a binary grammar, any rules in the original grammar that have more than two symbols on the right-hand side must be converted into a sequence of binary rules. The extra rules created in this process are called incomplete[1] rules. The topmost span in particular will usually need to be constructed in several steps, applying multiple incomplete rules before creating a complete span. If the grammar rules are always binarized from the left (right), then only cells on the left (right) edge of the chart can affect the top span; however, the grammar used in our parser is binarized "head-outward" (Klein and Manning, 2003), which means that potentially any cell in the chart can be used to create the top span.

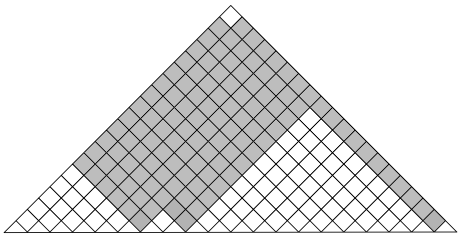


Figure 2: Example of a CKY chart with independence constraints. In the gray cells the modified algorithm will only loop over incomplete rules.

The combination of these two issues means that in order to correctly parse a sentence when an independent span boundary is missing from the input the modified CKY algorithm must process incomplete rules even at positions in the chart that cross a boundary. Thus in the modified algorithm, cells which do not cross an independent boundary are processed normally, and in cells which do cross a boundary the algorithm will avoid looping over complete binary rules.[2] Figure 2 shows an example CKY chart where boundary-crossing cells are colored gray.

3.1 How much work can we expect to save?

The core of the CKY algorithm is shown in Algorithm 1. For our purposes, we can consider the amount of work done in the CKY algorithm to be the number of binary edges visited in the inner loop (lines 4-10). For each cell the algorithm iterates over the binary rules in the grammar, calculating the probability of the left-hand-side at each split point. The number of these binary edges is

[1]E.g., if a rule $A \rightarrow BCD$ becomes $@_{BC} \rightarrow BC$ and $A \rightarrow @_{BC}D$, then the former is *incomplete* and the latter is *complete*.

[2]While boundary-crossing cells depend on non-crossing cells, the reverse is not the case; thus the non-crossing cells can all be processed before the crossing cells, or the cells can be looped over in the regular order, with a check inside the loop. While this may have implications for e.g. parallelization, we do not explore this idea further here.

```
for $1 \le i \le n$ do
    $T_{i,i+1} \leftarrow \{A|A \rightarrow a \in G \wedge w_i = a\}$
end
for $2 \le j \le n$ do
    for $1 \le i \le n - j + 1$ do
        for $i < k < i + j$ do
            $T_{i,i+j} \leftarrow \{A|A \rightarrow BC \in G \wedge B \in T_{i,k} \wedge C \in T_{k,i+j}\}$
        end
    end
end
```

Algorithm 1: The CKY algorithm. $T_{i,j}$ is the cell corresponding to words $w_i \dots w_{j-1}$.

Local Features	
t_{k-1}	t_k
t_{k-2}, t_{k-1}	t_k, t_{k+1}
$t_{k-3}, t_{k-2}, t_{k-1}$	t_k, t_{k+1}, t_{k+2}
Global Features	
t^l_i	$1 \le i < k - 1$
t^l_i, t^l_{i+1}	$1 \le i < k - 2$
$t^l_i, t^l_{i+1}, t^l_{i+2}$	$1 \le i < k - 3$
t^l_i	$k \le i < n - 1$
t^l_i, t^l_{i+1}	$k \le i < n - 2$
$t^l_i, t^l_{i+1}, t^l_{i+2}$	$k \le i < n - 3$

Table 1: Feature templates. k is the boundary position, t_k is the k th POS tag (level 0), and t^l_i is the i th POS tag in the l-level POS tag sequence.

$$|G| \left[\frac{n^3}{6} - \frac{n}{6} \right] \quad (1)$$

The amount of work saved depends on the number and locations of the independent span boundaries, as well as the proportion of complete rules in the grammar, denoted $\frac{|G_{comp}|}{|G|}$. We can consider two idealized scenarios: a) one boundary at $\frac{n}{K}$, and b) $K - 1$ boundaries at $\frac{n}{K}, \frac{2n}{K}, \dots, \frac{(K-1)n}{K}$, where K is an integer $1 < K < n$.

For the first case, the ratio of work saved approaches

$$\frac{|G_{comp}|}{|G|} \left[\frac{3}{K} - \frac{3}{K^2} \right] \quad (2)$$

as n grows. This limit converges quickly for $n \ge 10$. If we approximate $|G_{comp}|/|G|$ as 0.5 (for the grammar used by the parser in Section 5, it is $\approx .54$), then for $K = 2, 3, 4, \dots$, the values are $\frac{3}{8}, \frac{1}{3}, \frac{3}{32}, \dots$ Intuitively, for one boundary, the best location is exactly in the center of the sentence, and the upper limit on how much work is saved is about 37%.

For the case of $K-1$ boundaries equally spaced, the ratio is

$$\frac{|G_{comp}|}{|G|} \frac{K^2 - 1}{K^2} \quad (3)$$

The values for $K = 2, 3, 4, \dots$ are $\frac{3}{8}, \frac{4}{9}, \frac{15}{32}, \dots$ Clearly, the smaller pieces a sentence can be divided into the less work the parser will do; however, realistically most sentences will not have a large number of independent spans, and they will not be equal in length. We might take $K = 3$ as best-case estimate, giving us about 44%. Thus we can guess that a parser will be able to save around 35-45% of the work it does in the CKY algorithm loop by using independence constraints.

The derivations of Equations 1-3 are shown in Appendix A.

4 Classifying Independent Span Boundaries

In order to use independence constraints in a parser, we need to be able to identify boundaries between independent words in a sentence using only surface features (words and part-of-speech tags). We created a binary classifier which, given a POS-tagged sentence and a position between two words, decides whether those two words are independent or not. Our classifier currently uses only POS tags as features. We used `opal` (Yoshinaga and Kitsuregawa, 2010), a tool for fast online classification, to train and test the models, training on sentences from Penn Treebank section 02-21 and testing on section 22. We set opal to use the passive-aggressive perceptron update, and output probabilities in order to use a threshold to trade off precision and recall.

4.1 Features

We use only part-of-speech tags to create features for the classifier (adding lexical or other features is left to future work). The property of independence between two words is inherently global, as it can be affected by structure arbitrarily far away. Thus we have both local and global features. The global features are furthermore distinguished by **POS level**, explained in detail in the next section. The specific feature templates are shown in Table 1.

Features	#feats	Acc	Prec	Rec	F_1	$F_{0.5}$	TP	FP	FN	TN
p	37001	93.71	80.73	70.49	75.27	78.45	3679	878	1540	32320
P_0	33167	87.16	51.69	83.98	63.99	55.99	4383	4097	836	29101
p,P_0	70168	95.21	87.38	75.65	81.09	84.75	3948	570	1271	32628
p,P_1	37055	94.81	78.38	85.38	81.73	79.69	4456	1229	763	31969
p,P_2	39336	95.34	84.25	80.76	82.47	83.53	4215	788	1004	32410
p,P_3	46861	95.04	89.47	71.95	79.76	85.31	3755	442	1464	32756
p,P_0,P_1	70222	**95.48**	**88.95**	76.16	82.06	**86.06**	3975	494	1244	32704
p,P_0,P_2	72503	95.09	88.28	73.60	80.27	84.89	3841	510	1378	32688
p,P_0,P_3	80028	94.84	88.81	70.99	78.91	84.56	3705	467	1514	32731
p,P_1,P_2	39390	95.27	80.99	85.21	**83.04**	81.80	4447	1044	772	32154
p,P_1,P_3*	41553	**95.44**	**89.05**	75.74	81.86	**86.03**	3953	486	1266	32712
p,P_0,P_1,P_2,P_3	82417	95.35	86.89	77.49	81.92	84.83	4044	610	1175	32588

Table 2: Results of classifier using different combinations of features. (*Final feature configuration.)

Lvl0	Lvl1	Lvl2	Lvl3	Lvl0	Lvl1	Lvl2	Lvl3
NN	N	N	N	CD	X	X	#
NNP	N	N	N	-LRB-	X	X	B
NNPS	N	N	N	-RRB-	X	X	B
NNS	N	N	N	DT	X	X	D
PRP	N	N	N	PDT	X	X	D
VB	V	V	V	PRP$	X	X	D
VBD	V	V	V	WP$	X	X	D
VBG	V	V	V	JJ	X	X	J
VBN	V	V	V	JJR	X	X	J
VBP	V	V	V	JJS	X	X	J
VBZ	V	V	V	-RQ-	X	X	Q
,	X	,	,	-LQ-	X	X	Q
.	X	.	.	RB	X	X	R
:	X	:	:	RBR	X	X	R
CC	X	C	C	RBS	X	X	R
IN	X	I	I	EX	X	X	X
RP	X	I	I	FW	X	X	X
TO	X	T	T	LS	X	X	X
WDT	X	W	W	MD	X	X	X
WP	X	W	W	POS	X	X	X
WRB	X	W	W	SYM	X	X	X
#	X	X	#	UH	X	X	X
$	X	X	#				

Table 3: For each POS level, the original tag is replaced with the corresponding value.

4.2 POS Level

In previous unpublished work on a similar task, we found that heuristically transforming the POS tag sequence to create additional features can be beneficial. We refer to these transformations as **POS levels**. In this classifier we implemented three levels, in addition to the original POS tags as level 0.

We show all levels in Table 3. Each level specifies a value by which each level 0 tag is replaced during the transformation. The motivation behind each transformation is roughly as follows: level 1 is meant to capture clause nuclei; level 2 is further intended to show boundaries between clauses; and level 3 expands almost all the way back to the original tags, but with some distinctions erased, mostly to reduce the number of features.

4.3 Which Features Are Useful?

In order to find the best configuration of features for the classifier, and to evaluate the proposed POS levels, we tested the classifier using several different combinations. Selected results are shown in Table 2. In the "Features" column, p denotes the local features, and P_l denotes the global features from POS level l.

There are several things worth noting in these results. First, neither local nor global features are sufficient alone; it appears that local features promote precision, while global features promote recall. Second, examining the cases where global features are limited to a single POS level, it is apparent that each POS level has a different effect on precision and recall, thus confirming that the classifier is able to extract different signals from the different POS levels, as intended. Finally, combining all POS levels together actually reduces accuracy, possibly because the features are highly correlated (although see the discussion of the kernel classifier).

4.4 Results

To avoid degrading the accuracy of the parser as much as possible, we selected the feature configuration based on $F_{0.5}$ score, a measure which favors precision over recall. We chose p, P_1, P_3 over p, P_0, P_1 because the former had a slight edge in precision and fewer features.

More detailed results are shown in Table 4. We used a threshold on the score output by the classifier to reverse some of the classifier's decisions in a post-process step. Although it doesn't improve on the classifier in accuracy, the `precision` thresh-

Features	Threshold	Acc	Prec	Rec	F_1	$F_{0.5}$	TP	FP	FN	TN
p,P_1,P_3	default	95.44	89.05	75.74	81.86	86.03	3953	486	1266	32712
p,P_1,P_3	precision	94.99	91.65	69.44	79.01	86.14	3624	330	1595	32868
p,P_1,P_3	max precision	92.10	95.80	43.74	60.06	77.38	2283	100	2936	33098
p,P_1,P_3	recall	94.28	73.82	89.65	80.97	76.53	4679	1659	540	31539

Table 4: Results of linear classifier using different score thresholds.

Features	Threshold	Acc	Prec	Rec	F_1	$F_{0.5}$	TP	FP	FN	TN
p,P_0,P_1,P_2,P_3	default	97.47	92.17	88.91	90.51	91.50	4640	394	579	32804
p,P_0,P_1,P_2,P_3	precision	97.27	92.95	86.43	89.58	91.57	4511	342	708	32856
p,P_0,P_1,P_2,P_3	max precision	96.57	94.22	79.63	86.31	90.89	4156	255	1063	32943
p,P_0,P_1,P_2,P_3	recall	97.15	88.16	91.32	89.71	88.78	4766	640	453	32558

Table 5: Results of polynomial classifier using different score thresholds.

old did slightly improve in $F_{0.5}$.

4.5 Efficiency of the Classifier

The efficiency of the classifier is as important as the accuracy—it doesn't matter how much time is saved during parsing if it takes even longer to run the classifier. `opal` takes less than half a second to run on the instances from section 22; however, the instances are created by a Python script, which is not very optimized. This script takes about 100 seconds to run on the machine described in Section 5.1. While this time is already less than the time saved in the parser (see Section 5.2), it could be significantly reduced by re-implementing in Java or even C++. Thus the potential gains offered by this approach are not just theoretical.

4.6 Polynomial Kernel

For comparison with the linear classifier, we trained another classifier using a polynomial kernel (with degree 3) with all the features. The results are shown in Table 5. The polynomial kernel improves over the linear classifier in accuracy by 2%, in precision by 3 points, and in recall by just over 13 points. This suggests that there is a large potential for improving the linear classifier by adding conjunctive features. Alternatively, there are methods for effectively linearizing a kernel-based classifier, e.g. (Kudo and Matsumoto, 2003; Isozaki and Kazawa, 2002). Currently, the polynomial classifier takes over 2 hours to run on section 22 (training the model took almost 4 days).

5 Parsing With Independence Constraints

In order to demonstrate use of the independent constraints in a parser, we modified the CKY parser included in the Stanford Parser distribution to accept independent span boundaries as constraints and to use the modified CKY algorithm described above. Our modifications are:

- after reading in the grammar, index the incomplete binary rules
- read in the file containing the boundaries output by the classifier from the previous section
- for each CKY cell, if the cell spans a boundary then loop over just the incomplete binary rules
- if at the end of the CKY loop a parse was not successful, then loop again over just the cells which span a boundary and process all of the binary rules
- output the total number of times entering the inner loop as well as the number of times the parser failed

5.1 Experimental Setup

We used the modified Stanford Parser described above, with an unlexicalized grammar[3] extracted from the WSJ sections 02-21, and evaluated its performance on section 22 using output from the classifier as constraints. For the baseline, the

[3]The grammar was extracted using the Stanford Parser with command-line options `-acl03pcfg -noRebinarization -compactGrammar 1`

Parser	Time (s)	Speedup	# Binary Edges	F_1	Parse Failures
baseline	1558	-	1.75×10^{10} (100%)	85.85	0
linear	1283 (+100)	1.21× (1.12×)	1.08×10^{10} (62%)	83.71 (-2.14)	15
poly	1106 (+2h)	1.41× (.19×)	9.74×10^{09} (56%)	84.85 (-1.00)	6
oracle	1016	1.53×	8.47×10^{09} (48%)	86.71 (+0.86)	4

Table 6: Results of parsing with independence constraints. Results for both linear and polynomial classifiers are shown, as well as for the gold independent span boundaries. The times in parentheses are the classifier run times.

parser was given null constraints. The accuracies and times shown are those reported by the Stanford Parser.

All experiments were run on a DELL Precision 690, with 8 cores and 32G of RAM. Unless otherwise noted multiple processes were run in parallel, and times reported were not averaged over multiple runs. Since we saw significant variation of up to 10%, the times should be taken with a grain of salt. The computation done in the CKY algorithm is measured in the number of binary edges visited in the inner loop. A binary edge is a tuple of a span (begin & end), a binary rule $A \rightarrow BC$, and a split point (the position where B and C meet).

5.2 Results

The results of running the parser on section 22 using the linear classifier from Section 4.4 are shown in Table 6. The table shows the total time taken, the total times entering the inner loop, the F_1 and difference from the baseline, and the number of times the parse failed using the constraints. The parser with independence constraints saves 38% of the computation inside the CKY loop over the baseline, corresponding to about 20% reduction in total parse time (12% if the running time of the classifier is included), at the cost of a 2-point drop in F-score. Detailed results of further experiments with various thresholds on sentence length and classifier score are shown in Appendix B.

5.3 Polynomial Kernel

A difference of 2 F_1 score is not small, but on the other hand it is about by how much the unlexicalized Stanford Parser trails the Collins parser, for example. However, as shown above in Section 4.6, there is room to improve the linear classifier through conjunctive features. As an indication of an upper bound of the achievable performance, we tried using the output of the kernel classifier in the parser as above, while acknowledging that at present the time needed to produce the classifier output dwarfs the time needed to actually parse the test data.

Parser	Time (s)	Speedup	F_1
baseline	1538		85.54
linear	1106 (+100)	1.39× (1.28×)	83.55 (-1.99)
poly	1040	1.48×	84.57 (-0.97)

Table 7: Results of parsing WSJ section 23.

The results of running the parser on section 22 with the polynomial classifier output are shown with the previous results in Table 6. With the more accurate classifier, the parser is able to reduce the necessary computation even further, by 44%, while losing less accuracy.

5.4 Gold Independent Span Boundaries

For another comparison, we tested the parser using the gold independent span boundaries. The results for section 22 are shown in Table 6. The number of binary edges visited is cut in half, and parsing accuracy is improved by almost 1 point. It is interesting to note that the parser was unable to parse 4 sentences with the gold constraints (the grammar only allowed a parse that violated the gold boundaries).

5.5 WSJ Section 23

To compare with previous work on parsing using the Penn Treebank, we show the time and accuracy for parsing section 23, using both linear and kernel classifier output along with the baseline parser in Table 7. The times reported are the average of three runs each. Because there was significant variation in parse time when multiple processes were run in parallel, for these results only one process was run at a time. The results parallel those shown on the development data.

As a point of comparison, Roark et al. (2012) reported speedups of 1.6-2x with no loss of accuracy. These results are not directly comparable due to differences in parser (their parsers use beam search variants of CKY and coarse-to-fine pruning) and grammar (they used the Berkeley latent variable grammar and a lexicalized grammar).

6 Related Work

There are several strains of research related to adding constraints to the CKY chart. (Roark et al., 2012) describes an approach using finite-state taggers to decide whether each word in a sentence begins or ends a multiword constituent and has a unary span or not. They show that their tagger is able to achieve very high precision, reducing parse time without negatively affecting accuracy.

(Bodenstab et al., 2011) proposes a classifier which directly decides for each cell in the chart how many constituents should be created. Their parser uses beam search with a FOM and a beam for each chart cell.

Like these approaches, our method uses a classifier to avoid doing work in certain chart cells. While not completely orthogonal, we believe our independence constraints are complementary. A single decision by our classifier closes a large swath of cells based on the global structure, while their methods make local decision using local information. The high accuracy of their classifiers shows the necessity of improving our model.

(Yarmohammadi et al., 2014) proposes a concept of ‘hedge’ parsing, where only spans below a certain length are allowed, and show how this reduces the computation done by the CKY algorithm. Their system does not create spans of length larger than the threshold and thus doesn’t follow the original treebank annotation, while our approach is able to return the original gold parse tree, provided that the classifier does not output a false positive. Their approach of segmenting a sentence before parsing is essentially the same as ours, but they segment based on a maximum span length and their classifier is based on a finite-state sequence model.

There is some previous research using clause boundaries to constrain dependency parsers (Ohno et al., 2006; Husain et al., 2011; Kim and Lee, 2004). This is more linguistically motivated than our constraints; indeed, the approaches appear to rely on processing specific to each language. It is difficult to compare with these results directly; however, although only (Ohno et al., 2006) reported parse times, all three papers reported improved accuracy.

7 Conclusions

We have proposed a property of **independence** between words in a sentence, and shown how to use this property to create top-down constraints which can be used to reduce the computation done by the CKY algorithm. We demonstrated two classifiers for identifying boundaries between independent words given a sentence with only surface features, a linear classifier which is fast but less accurate, and a classifier with a polynomial kernel which is much more accurate but very slow. We then showed a significant improvement in speed over a strong baseline CKY parser by using the output of these classifiers to create top-down constraints at the cost of some accuracy.

Although the loss of accuracy when using the linear classifier is currently uncomfortably large, there are several possible avenues for improvement. The performance of the kernel classifier indicates that there is room for improvement by manually adding conjunctive features to the linear classifier or using a method to automatically linearize the model. Features based on words as well as POS tags may also be beneficial. Changing the model itself to, e.g., a sequence model might also help. However, the current approach has several weaknesses which should be addressed by future research.

First, the top-down nature of the independence constraints does not make a natural fit with the bottom-up CKY algorithm. In particular, the presence of incomplete rules in the grammar combined with the bottom-up search means that the parser still ends up doing some computation to create spans which violate the constraints, even though it is prevented from completing such a span.

Second, the pipelined nature of the classifier means that it only has access to POS tags and in particular is not able to make use of information generated as the parser processes lower-level spans. Tighter integration of the classifier into the parser may be beneficial to both.

Third, the current classifier combines instances from different syntactic structures into a single model. It is possible that training multiple models on different types of sentences would result in a better classifier.

References

Srinivas Bangalore and Aravind K Joshi. 1999. Supertagging: An Approach to Almost Parsing. *Computational Linguistics*, 25:237–265.

Nathan Bodenstab, Aaron Dunlop, Keith Hall, and Brian Roark. 2011. Beam-Width Prediction for Efficient Context-Free Parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, pages 440–449, Portland, Oregon. Association for Computational Linguistics.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *ACL 2005*.

Jason Eisner and Giorgio Satta. 1999. Efficient parsing for bilexical context-free grammars and head automaton grammars. In *ACL 1999*.

Samar Husain, Phani Gadde, Joakim Nivre, and Rajeev Sangal. 2011. Clausal parsing helps data-driven dependency parsing: Experiments with hindi. In *IJCNLP 2011*, pages 1279–1287. The Association for Computer Linguistics.

Hideki Isozaki and Hideto Kazawa. 2002. Efficient support vector classifiers for named entity recognition. In *COLING 2002*.

Mi-Young Kim and Jong-Hyeok Lee. 2004. Syntactic analysis of long sentences based on s-clauses. In Keh-Yih Su, Jun'ichi Tsujii, Jong-Hyeok Lee, and Oi Yee Kwong, editors, *IJCNLP 2004*, volume 3248 of *Lecture Notes in Computer Science*, pages 518–526. Springer.

Dan Klein and Christopher D Manning. 2003. Accurate Unlexicalized Parsing. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 423–430, Sapporo, Japan, July. Association for Computational Linguistics.

Taku Kudo and Yuji Matsumoto. 2003. Fast methods for kernel-based text analysis. In *ACL 2003*, pages 24–31.

Tomohiro Ohno, Shigeki Matsubara, Hideki Kashioka, Takehiko Maruyama, and Yasuyoshi Inagaki. 2006. Dependency parsing of japanese spoken monologue based on clause boundaries. In *ACL 2006*. The Association for Computer Linguistics.

Slav Petrov and Dan Klein. 2007. Improved inference for unlexicalized parsing. In *Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, April 22-27, 2007, Rochester, New York, USA*, pages 404–411.

Brian Roark, Kristy Hollingshead, and Nathan Bodenstab. 2012. Finite-State Chart Constraints for Reduced Complexity Context-Free Parsing Pipelines. *Computational Linguistics*, 38(4):702–753.

Kentaro Torisawa, Kenji Nishida, Yusuke Miyao, and Jun'ichi Tsujii. 2000. An HPSG parser with CFG filtering. *Natural Language Engineering*, 6(1):63–80.

Mahsa Yarmohammadi, Aaron Dunlop, and Brian Roark. 2014. Transforming trees into hedges and parsing with "hedgebank" grammars. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA*, pages 797–802. The Association for Computer Linguistics.

Naoki Yoshinaga and Masaru Kitsuregawa. 2010. Kernel Slicing: Scalable Online Training with Conjunctive Features. In Chu-Ren Huang and Dan Jurafsky, editors, *Proceedings of the 23rd International Conference on Computational Linguistics COLING 2010*, pages 1245–1253, Beijing. Tsinghua University Press.

Appendix A. Derivation of equations in section 3.1

The amount of computation done in lines 4-10 of Algorithm 1 can be calculated as follows:

$$\begin{aligned}
&\sum_{j=2}^{n}\sum_{i=1}^{n-j+1}\sum_{k=i+1}^{i+j-1}|G| \\
=&|G|\sum_{j=2}^{n}(n-j+1)(j-1) \\
=&|G|\sum_{i=1}^{n-1}(n-i)(i) \\
=&|G|\sum_{i=1}^{n-1}(ni-i^2) \\
=&|G|(n\sum_{i=1}^{n-1}i-\sum_{i=1}^{n-1}i^2) \\
=&|G|(n\frac{(n-1)n}{2}-(\frac{n^3}{3}-\frac{n^2}{2}+\frac{n}{6})) \\
=&|G|(\frac{1}{2}n^3-\frac{1}{2}n^2-\frac{1}{3}n^3+\frac{1}{2}n^2-\frac{1}{6}n) \\
=&|G|(\frac{1}{6}n^3-\frac{1}{6}n)
\end{aligned}$$

This is the number of binary edges evaluated by the CKY algorithm. Using independence constraints, the algorithm avoids doing any computation for complete edges in spans which violate the constraints. The work saved is thus the number of complete binary edges in the entire chart minus the number of complete edges that are actually processed in cells that satisfy the constraints. For a single independent boundary at $\frac{n}{K}$, we get:

$$\begin{aligned}
&|G_{comp}|[\frac{1}{6}n^3-\frac{1}{6}n] \\
&-|G_{comp}|[\frac{1}{6}(\frac{n}{K})^3-\frac{1}{6}\frac{n}{K}] \\
&-|G_{comp}|[\frac{1}{6}(\frac{(K-1)n}{K})^3-\frac{1}{6}\frac{(K-1)n}{K}] \\
=&|G_{comp}|[\frac{1}{6}n^3-\frac{1}{6}n-\frac{1}{6}\frac{(K-1)^3+1}{K^3}n^3+\frac{1}{6}n] \\
=&|G_{comp}|[\frac{1}{6}\frac{K^3}{K^3}n^3-\frac{1}{6}\frac{K^3-3K^2+3K}{K^3}n^3] \\
=&|G_{comp}|\frac{3K^2-3K}{6K^3}n^3
\end{aligned}$$

The proportion of work saved relative to the original algorithm is then

$$\frac{|G_{comp}|\frac{3K^2-3K}{6K^3}n^3}{|G|(\frac{1}{6}n^3-\frac{1}{6}n)}$$

which depends on n as well as K; however, we can approximate this as the limit as n goes to infinity:

$$\begin{aligned}
&\lim_{n\to\infty}\frac{|G_{comp}|\frac{3K^2-3K}{6K^3}n^3}{|G|(\frac{1}{6}n^3-\frac{1}{6}n)} \\
=&\frac{|G_{comp}|}{|G|}[\frac{3}{K}-\frac{3}{K^2}]
\end{aligned}$$

Similarly, the work saved with K evenly-spaced boundaries is

$$\begin{aligned}
&|G_{comp}|[\frac{1}{6}n^3-\frac{1}{6}n] \\
&-K|G_{comp}|[\frac{1}{6}(\frac{n}{K})^3-\frac{1}{6}\frac{n}{K}] \\
=&|G_{comp}|[\frac{1}{6}n^3-\frac{1}{6}n-\frac{1}{6}\frac{1}{K^2}n^3+\frac{1}{6}\frac{n}{K}] \\
=&|G_{comp}|\frac{1}{6}\frac{K^2-1}{K^2}n^3
\end{aligned}$$

and the proportion of the original work saved is approximately

$$\begin{aligned}
&\lim_{n\to\infty}\frac{|G_{comp}|\frac{1}{6}\frac{K^2-1}{K^2}n^3}{|G|(\frac{1}{6}n^3-\frac{1}{6}n)} \\
=&\frac{|G_{comp}|}{|G|}\frac{K^2-1}{K^2}
\end{aligned}$$

Appendix B. Detailed parse results

We experimented with a post-processing step to adjust the recall and precision of the classifier, as well as adding a threshold on the minimum length of a sentence to apply constraints to in the parser (on the hypothesis that longer sentences are likely to gain a proportionally larger advantage). We show the detailed results from the parser in Table 8, using both the linear and polynomial classifiers. Sentences shorter than `MinSentLen` were parsed without constraints.

The results are largely as expected. Sentences less than 20 words do not affect the results much. The `recall` threshold predictably results in a large loss in classifier precision and thus parsing accuracy. We note the results in boldface: with a high precision threshold, the polynomial classifier is able to reduce the computation in the CKY loop by 42% while losing less than half a point in F_1 score.

Classifier	**MinSentLen**	**Constraints**	**Time (s)**	**# Edges**	**F_1**	**Parse Failures**
-	-	baseline	1558	$1.75{\times}10^{10}$ (100%)	85.85	0
linear	0	default	1283	$1.08{\times}10^{10}$ (62%)	83.71 (-2.14)	15
linear	0	precision	1143	$1.13{\times}10^{10}$ (65%)	84.05 (-1.80)	7
linear	0	max precision	1384	$1.42{\times}10^{10}$ (81%)	85.55 (-0.30)	2
linear	0	recall	1024	$7.80{\times}10^{09}$ (45%)	78.74 (-7.11)	136
linear	20	default	1126	$1.12{\times}10^{10}$ (64%)	84.17 (-1.68)	9
linear	20	precision	1313	$1.16{\times}10^{10}$ (66%)	84.43 (-1.42)	4
linear	20	max precision	1338	$1.44{\times}10^{10}$ (82%)	85.59 (-0.26)	2
linear	20	recall	1121	$8.24{\times}10^{09}$ (47%)	80.38 (-5.47)	103
linear	30	default	1312	$1.28{\times}10^{10}$ (73%)	84.82 (-1.03)	3
linear	30	precision	1279	$1.31{\times}10^{10}$ (75%)	85.01 (-0.84)	1
linear	30	max precision	1485	$1.53{\times}10^{10}$ (87%)	85.63 (-0.22)	1
linear	30	recall	1140	$1.02{\times}10^{10}$ (58%)	82.79 (-3.06)	57
linear	40	default	1476	$1.51{\times}10^{10}$ (86%)	85.56 (-0.29)	1
linear	40	precision	1390	$1.52{\times}10^{10}$ (87%)	85.59 (-0.26)	0
linear	40	max precision	1513	$1.65{\times}10^{10}$ (94%)	85.75 (-0.10)	0
linear	40	recall	1403	$1.33{\times}10^{10}$ (76%)	84.65 (-1.20)	14
poly	0	default	1106	$9.74{\times}10^{09}$ (56%)	84.85 (-1.00)	6
poly	0	precision	1118	$9.84{\times}10^{09}$ (56%)	85.12 (-0.73)	4
poly	0	max precision	1137	**$1.02{\times}10^{10}$ (58%)**	**85.42 (-0.43)**	2
poly	0	recall	1050	$9.25{\times}10^{09}$ (53%)	84.05 (-1.80)	33
poly	20	default	1070	$1.02{\times}10^{10}$ (58%)	85.08 (-0.77)	5
poly	20	precision	1172	$1.03{\times}10^{10}$ (59%)	85.25 (-0.60)	3
poly	20	max precision	1092	$1.06{\times}10^{10}$ (61%)	85.41 (-0.44)	2
poly	20	recall	1088	$9.68{\times}10^{09}$ (55%)	84.75 (-1.10)	7
poly	30	default	1222	$1.20{\times}10^{10}$ (69%)	85.57 (-0.28)	1
poly	30	precision	1267	$1.20{\times}10^{10}$ (69%)	85.62 (-0.23)	1
poly	30	max precision	1238	$1.23{\times}10^{10}$ (70%)	85.65 (-0.20)	1
poly	30	recall	1238	$1.16{\times}10^{10}$ (66%)	85.44 (-0.41)	2
poly	40	default	1465	$1.49{\times}10^{10}$ (85%)	85.72 (-0.13)	0
poly	40	precision	1353	$1.49{\times}10^{10}$ (85%)	85.75 (-0.10)	0
poly	40	max precision	1570	$1.50{\times}10^{10}$ (86%)	85.78 (-0.07)	0
poly	40	recall	1489	$1.47{\times}10^{10}$ (84%)	85.69 (-0.16)	1

Table 8: Results from parsing section 22 using constraints from both linear and polynomial classifiers, varying minimum sentence length and classifier probability threshold.

Dependency Parsing with Compression Rules

Pablo Gamallo
Centro Singular de Investigación en Tecnoloxías da Información - CITIUS
University of Santiago de Compostela, Galiza, Spain
pablo.gamallo@usc.es

Abstract

This article proposes a syntactic parsing strategy based on a dependency grammar containing both formal rules and a *compression* technique that reduces the complexity of those rules. Compression parsing is mainly driven by the single-head constraint of Dependency Grammar, and can be seen as an alternative method to the well-known *constructive* strategy. The compression algorithm simplifies the input sentence by progressively removing from it the *dependent* tokens as soon as binary syntactic dependencies are recognized. The performance of our system was compared to a deterministic parser based on supervised learning: MaltParser. Both systems were applied on several test sets of sentences in Spanish and Portuguese, from a variety of different domains and genres. Results showed that our parsing method keeps a similar performance through related languages and different domains, while MaltParser, as most supervised methods, turns out to be very dependent on the text domain used to train the system.

1 Introduction

For large scale applications in Information Extraction (IE), syntactic parsing should be robust, fast, and relatively accurate. Moreover, for specific IE applications such as semantic relation extraction, the output of parsing should be simple, easy to handle by the IE systems, and close to the semantic relationships to be extracted. For multilingual purposes, it is important to develop parsing techniques easily adapted to several languages. And finally, in order to be easily integrated in several NLP applications and tasks, the parsers should be applied on different text domains and genres with similar accuracy.

There are two well known approaches that could be considered as good approximations to the ideal system filling all these parsing properties: both deterministic dependency parsing (Nivre, 2004) and partial parsing using rule-based finite-state techniques (Abney, 1996). However, these two parsing approaches have still some problems.

Recent work on deterministic dependency parsing (called 'transition based') relies on supervised techniques requiring fully analyzed training corpora. Given that supervised techniques tend to have loss of precision when applied on texts of domains and genres different to those used for training (Rimell et al., 2009; Gildea, 2001), they need too much manual effort to create, adapt, or modify the training corpus to the target domain.

Speed is not actually a problem for finite-state techniques, which can parse large text corpora in a very efficient way. However, as they rely on complex rule-based notations, their main drawback is the difficulty to adapt such a rule system to different languages. Moreover, as most finite-state parsers are based on constituency grammars, their syntactic output cannot be easily integrated into IE applications. Unlike phrase constituents, dependencies are seen as simple and flat syntactic representations, very close to semantic relations which are the extraction target of many IE systems.

Many finite-state parsers are based on the *constructive* strategy (Grefenstette, 1996; Abney, 1996; Ait-Mokhtar and Chanod, 1997). In constructive parsing, an input sentence is manipulated by transducers that progressively transform the input with additional symbols encoding syntactic constituents or dependency relations (Oflazer, 2003). These transducers are pattern rules arranged in cascades: the output of a transducer is the input of the next one, which contains new rules adapted to the symbols added to the input.

So, parsing consists in transforming a basic input string into a more complex one by incrementally adding new symbols, bracket delimiters, labels, or special markers. This strategy incrementally constructs the linguistic representation within the input string, by making use of rules organized at different levels of complexity.

In this article, we propose a new (rule-based) finite-state parsing strategy based on dependencies, which minimizes the complexity of rules by using a technique we call *compression*. Compression parsing is driven by the ”single-head” constraint of Dependency Grammar, and can be seen as an alternative method to the *constructive* strategy. It simplifies the input string by progressively removing the *dependent* tokens as binary syntactic dependencies are recognized. At the end of the compression process, if all the dependencies in the sentence were recognized, the input string should contain just one token representing the main head (i.e., the *root*) of the sentence. This strategy was inspired by the *Right* and *Left* Reduce transitions used in deterministic dependency parsing. The input sentence is assumed to be tagged and disambiguated with a Part-Of-Speech (PoS) tagger. Moreover, the cost of manually creating rules can also be reduced by providing a suitable rule notation for linguists.

As in Ait-Mokhtar and Chanod (1997), we hold that the ordering of rules/transducers is in itself a genuine linguistic task, which must preserve the basic principle of doing the easiest task first (Abney, 1996). If rules (and so the grammar) are written following this principle, it is possible to use finite-state automata to deal with embedding structures and long-distance dependencies. Note that this is a deterministic parsing strategy, since it cannot produce ambiguous structures. The use of grammars in recent dependency parsers is almost non-existent. Our work propose to incorporate more linguistic knowledge into the parsing systems *via* light-weight grammars.

Finally, a system based on the compression strategy was implemented and released under General Public License. In addition, we defined a high level grammar language to define dependency-based rules and developed a grammar compiler to generate compression parsers in several languages (Gamallo and González, 2011). The performance of this system was compared to MaltParser (Nivre et al., 2007b), a deterministic parser based on supervised learning. Both systems were applied on several test sets of sentences of different domains and genres, in Spanish and Portuguese. One of the main motivations of the evaluation is to test whether the two systems are reliable to parse sentences of different domains and genres. It is generally accepted that supervised classifiers require some type of domain adaptation when both the training and test data sets belong to different domains. In particular, the accuracy of statistical parsers degrades when they are applied to different genres and domains (Rimell et al., 2009; Gildea, 2001). Results showed that our system keeps a similar performance through related languages and different domains, while MaltParser, as most supervised methods, is very dependent on the text domain used to train the system.

The remainder of this article is organized as follows. Section 2 introduces different approaches on both dependency and FST parsing. Then, Section 3 is focused on the description of our compression strategy. Next, Section 4 provides a general view of the implemented system. Then Section 5 reports the diverse experiments performed over the Portuguese and Spanish data sets. And finally, some conclusions are addressed in Section 6.

2 Related Work

Our strategy is based on both dependencies and FST parsing.

2.1 Dependency-based Syntactic Parsing

Following Nivre (2005), there are two traditions in dependency parsing: grammar-driven and data-driven parsing. Within each tradition, it is also possible to distinguish between two different approaches: non-deterministic and deterministic parsing. In the latest years, many works on dependency parsing have been developed within the approach to data-driven deterministic parsing, which is also known as *transition-based parsing*, in opposition to grammar-driven parsers. Other data-driven strategies are non-deterministic such as *graph-based dependency parsing* (McDonald and Pereira, 2006; Carreras, 2007; Martins et al., 2010) .

Transition parsing consists in inducing statistical models in combination with a deterministic strategy based on shift-reduce parsing (Nivre, 2004; Yamada and Matsumoto, 2003; Gómez-Rodríguez and Fernández-González, 2012). In

Nivre et al. (2004), the parsing strategy uses the arc-eager algorithm. In Gómez-Rodríguez et al. (2014), this algorithm is simplified by just using two transitions on undirected dependencies (the head-dependent information is erased), so as to avoid error propagation.

As we will show later, the main problem of the supervised learning strategies arises when the test sentences belong to linguistic domains very different from those found in the training corpus. We will show later the negative effect on system performance when the test and training data sets does not belong to the same domain and genre. As hand labeling data in new domains is a costly enterprise, the domain adaptation problem is a fundamental challenge in machine learning applications. Note that many NLP annotated resources are based on text from the news domain (in most cases, the Wall Street Journal), which is a poor match to other domains such as biomedical texts, electronic mails, transcription of meetings, administrative language, etc. (Daumé-III and Marcu, 2007; Daumé-III, 2006).

2.2 Finite-State Parsing Techniques

Finite-state technology has attractive properties for syntactic parsing, such as conceptual simplicity, flexibility, and efficiency in terms of space and time. It allows to build robust and deterministic parsers. Most finite-state based parsing strategies use cascades of transducers and are known as *constructive* parsers.

Parsing based on cascades of finite-state transducers can be viewed as a sort of string transformation. Finite-state transducers introduce progressively markings and labels within the input text. Transducers are arranged in cascades (or layers), where the subsequent transducer takes the output of the previous one as input. After a certain number of cascades, the initial input (which is a tagged sentence) is transformed into a structured text enriched with syntactic marks, such as chunk boundaries, labels for heads, special markers for functions or for relations between heads, etc. This strategy, known as *constructive*, progressively constructs the linguistic representation within the input string, by making use of rules/transducers organized at different levels (or layers) of complexity.

Most of finite state strategies aim to construct, not dependency graphs, but phrase based structures (Ait-Mokhtar et al., 2002; Ciravegna and Lavelli, 2002; Kokkinakis and Kokkinakis, 1999; Ait-Mokhtar and Chanod, 1997; Abney, 1996; Joshi, 1996; Grefenstette, 1996). In general, the construction of these structures is performed with three main cascades/layer of rules: chunking, head recognition, and attachment. The first layers of rules transform the tagged input into sequences of symbols representing basic chunks. Then, further rules take those chunks as input to add new symbols marking the heads of each chunk and, finally, new rules are applied on the output of the previous ones to annotate the identified heads with labels of syntactic functions (attachment).

The number of finite-state approaches focused on constructive dependency parsing is much smaller. We can merely cite the work by Oflazer (2003), where the input string is progressively enriched by additional symbols encoding dependency relations between words.

The finite-state strategy often relies on one fundamental property: *easy-first parsing*. Easy-first parsing means that the simplest tasks must be done first, leaving the harder decisions for the last steps of the parsing process. Parsing proceeds by growing *islands of certainty* (Abney, 1996; Eisner and Smith, 2010; Goldberg and Elhadad, 2010; Tratz and Hovy, 2011; Versley, 2014).

Finite-state parsers are the fastest systems among those achieving linear time complexity. So, they are scalable as the input text increases in size and are easily integrated into IE applications exploring the Web as corpus.

3 A Compression Parsing Strategy

We propose yet another FST based method, very similar to the constructive approaches, but by making use of a similar strategy to the shift-reduce algorithm as in incremental parsing. We call it *compression parsing*. It consists of a set of transducers/rules that *compress* the input sequence of tokens by progressively removing the dependent tokens as soon as dependencies are recognized. So, at each application of a rule, the systems reduce the input and make it easier to find new dependencies in further rule applications. In particular, short dependencies are recognized first and, as a consequence, the input is simplified so as to make lighter the recognition of long distance dependencies. This is inspired by the easy-first strategy.

The input of our parsing method is a sequence of disambiguated tagged tokens, where each token is associated with two pieces of information: a PoS tag representing the basic morpho-syntactic category of the token (NOUN, VERB, PRP, etc.) and a feature structure containing other relevant information of the token: morphological information (number, tense, person, etc.), lemma, token string, token position, etc. Tagged tokens are the elementary objects of the parsing process, while rules, which are implemented as finite state transducers, operates on tagged tokens. More precisely, rules successively identify dependencies between tokens, remove the dependents (if required) from the input sequence, and update (if required) the feature structures of the heads.

3.1 Description of rules

A rule is a tuple $< P, Arc, \triangle, Reduce >$, where:

- P is a pattern of tagged tokens, defined as a regular expression, whose general form is $\alpha X \beta Y \gamma$. X and Y are non-empty strings representing two tagged tokens, considered as the *core elements* of the pattern; while α, β, and γ represent the left, middle, and right contexts, respectively, of the core elements; they may be empty.

- Arc is the action that creates a dependency link between the core elements (X and Y), when a subsequence of the tagged input is matched by the pattern; two types of arcs are distinguished: $Left\_Arc$ adds a dependency link between X and Y, being X the dependent and Y the head; $Right\_Arc$ creates a dependency link between X and Y, being X the head and Y the dependent. This action also assigns a label (subject, modifier, adjunct, etc) to the dependency.

- $\triangle$ is a set of operations (*Agreement*, *Add*, *Correction*, *Inherit*, etc.) that are applied on the feature structure of the core elements; they can be used to perform very different actions: verifying if the two core elements (i.e., head-dependent) share a set of feature-values, adding new feature-values to the head, modifying some values of the head, correcting PoS tags, allowing the head to inherit selected values from the dependent, etc. *Add* and *Inherit* can be seen as constructive operations.

- $Reduce$ is the action that removes the *dependent* from the input string; this action can be suspended if the dependent token is likely to be the head in other dependencies that has not been recognized yet. So, the dependent will not be reduced until all its potential dependent tokens have been recognized.

Compressing rules, not only reduce the complexity of the search space (or input) of the remaining rules to be applied, but also construct relevant information (by adding linguistic features) for the application of those rules. In particular, it may store in the head relevant information of the removed dependent (*Inherit* operation), it permits to generate new attributes or modify values from existing attributes (*Add* operation), and also it can correct odd tagged PoS tags (Correction) (Garcia and Gamallo, 2010). In sum, the main contribution of our work is to define compressing rules as the integration of two parsing techniques: both transition-based and constructive parsing. On the one hand, the rules reduce the search space by removing the dependent tokens and, on the other hand, they can add relevant information to the head tokens by making use of operations such as *Add* or *Inherit*. Rules compresses the input sequence of tokens so as to make it easier the identification of more distant dependencies.

The *Inherit* operation is one of the most innovative contributions of our rule system. It allows transferring linguistic features to heads before removing the dependent tokens from the search space. This can be considered as one of the main contributions of our dependency-based strategy. In combination with *Add* operation, *Inherit* is useful to transfer relevant information from auxiliary, light, or modal verbs to their main verbs. It can also be used to model coordination by transfering categorial information from the coordinated structures to the coordinator, so as to make it possible subject-verb agreements. For instance, in the sentence ‘Paul and Mary are eating’, the *Inherit* operation allows the coordinator ‘and’ to inherit the nominal category of their parts and, by means of the *Add* operation, we can assign it the *plural* number. In addition, *Inherit* can also be used to transfer relevant morphological information (third person, plural, present tense) to the root verb ‘eat’ from the auxiliar ‘are’. This way, there is grammatical agreement between ‘(are) eating’ (3rd person and plural) and its subject ‘Paul and Mary’. As far as we know, no dependency grammar/parser has proposed this solution to the dependency-based analysis of verbal periphrases and coordinations.

3.2 Rule Ordering: Easy First

As was mentioned above, the ordering of rules in a FST parser is a genuine linguistic task. Rules are ordered in such a way that the easiest tasks, for instance short dependencies, are performed first. As in (Eisner and Smith, 2010; Goldberg and Elhadad, 2010; Tratz and Hovy, 2011; Versley, 2014), we assume that correct parsers exhibit a short-dependency preference: a word's dependents tend to be close to it in the string. The fact of identifying first easy syntactic structures, such as those ruled by modifiers and specifiers, allows us to easily find later distant links, for instance those relating verbs with subordinate conjunctions. Let us take the expression: ‘if the former president of USA says ...’. There is here a long distance dependency between the verb ‘says’ and the conditional conjunction ‘if’. In most sentences, both words are not adjacent since a great variety of tokens can be interposed. However, in a compression approach, we can guess that dependency by making use of a very simple pattern consisting in a subordinate conjunction (type:S) appearing immediately to the left of a verb (last rule T_6 below in 1). We just need the following sequence of transductions/rules[1]:

$$
\begin{array}{ll}
T_1: & \texttt{PRP} \longleftarrow \texttt{PRP} \quad \texttt{NOUN} \\
T_2: & \texttt{NOUN} \longleftarrow \texttt{ADJ} \quad \texttt{NOUN} \\
T_3: & \texttt{NOUN} \longleftarrow \texttt{DT} \quad \texttt{NOUN} \\
T_4: & \texttt{NOUN} \longleftarrow \texttt{NOUN} \quad \texttt{PRP} \\
T_5: & \texttt{VERB} \longleftarrow \texttt{NOUN} \quad \texttt{VERB} \\
T_6: & \texttt{VERB} \longleftarrow \texttt{CONJ<type:S>} \quad \texttt{VERB}
\end{array} \tag{1}
$$

In Figure 1, we show the application of the six rules on the input expression, as well as the effect of the *Reduce* transition at each level (for the sake of simplicity, label assignment is not taken into account).

Each rule processes the input from left to right repeatedly as long as new dependencies satisfying the pattern are found. Rules are checked top-down following the rank imposed by the linguist. When

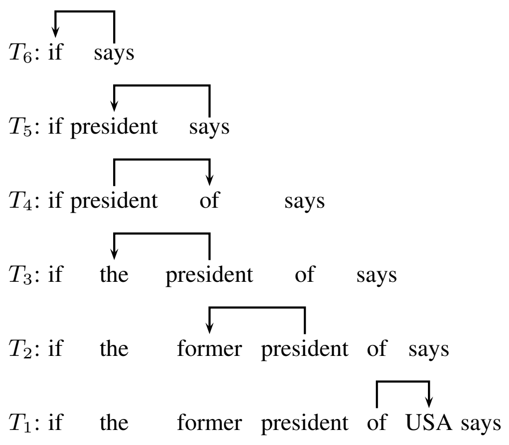

T_6: if says
T_5: if president says
T_4: if president of says
T_3: if the president of says
T_2: if the former president of says
T_1: if the former president of USA says

Figure 1: The six levels of analysis of ‘if the former president of USA says...’

the parser reaches the last rule of the ranked list, if at least one dependency has been identified, the parser starts again from the beginning until no new dependency is found. So, the parser works iteratively until no change is made.

4 The Implementation

4.1 The Modules

Our compression parsing strategy has been inserted into a more generic natural language architecture, which consists of the following modules:

A set of PoS tagging *adapters* that modify the output of two PoS taggers, namely FreeLing (Padró and Stanilovsky, 2012) and Tree-Tagger (Schimd, 1995), so as to generate an unified PoS tagged format. The result of this process is the input of compression parsers. A set of grammars written with a specific grammar notation. The grammar formalism was described in (Gamallo and González, 2011). A grammar compiler, written in Ruby, that takes a particular grammar as input and generates a compression parser, written in Perl. An a set of multilingual parsers, generated by the compiler from various grammars.

The whole system, called DepPattern, is released under the GNU General Public License (GPL). [2]. Five parsers were generated for the following languages: Spanish, Portuguese, English, French, and Galician. The parsers are robust and very efficient: they are able to parse about 3,000

[1]To simplify, rule notation is focused on just the final *Reduce* operation

[2]`http://gramatica.usc.es/pln/tools/deppattern.html`

words per second on a processor Core 2 Quad, 2.8 Ggz. The system can be run on any GNU/Linux distribution. DepPattern was used for several web-based IE applications, namely Open Information Extraction from Wikipedia (Gamallo et al., 2012), extraction of semantic relations with distant supervision (Garcia and Gamallo, 2011), and extraction of bilingual terminologies from comparable corpora (Gamallo and Pichel, 2008). It has also been integrated into commercial tools, e.g. Linguakit[3] and Avalingua[4].

4.2 Multilingual Parsing

The parsers were developed for five languages: Spanish, Portuguese, Galician, French, and English. However, we have just written two small, and not very different, grammars:

Romance Grammar A grammar with 96 rules that are all applied to four Latin languages, namely, Spanish, French, Galician, and Portuguese.

English Grammar A grammar with 104 rules for English.

The cost of writing these two grammars is quite low. They are small and almost identical, since they share most rules except a reduced group of them which is specific for English. The strategy we followed to write grammars is based on two methodological principles: to start with rules with high coverage, and to start with rules shared by as many languages as possible

The objective is to find a trade-off between high performance and low effort, i.e. we look for efficiency. Most DepPattern rules of our grammars satisfy these two principles, giving rise to broad-coverage parsers. The quality of the French grammar is not good enough since this Latin language is quite different from Spanish, Portuguese, and Galician. There are important rules specific for French which have not been integrated in the shared grammar, for instance rules for dealing with partitives. Besides, we have not defined non-projective rules since, in general, they have low coverage and are language-dependent. Finally, the grammars also contain lexicalized rules for prepositions, modal verbs, quantifiers, etc. However, as the sets of lexical units are declared in external configuration files, the grammars are not required to be modified.

[3]`https://linguakit.com/`

[4]`http://cilenis.com/en/avalingua/`

5 Experiments

Our objective is to compare the performance of our FST-based strategy with that of a transition-based system. For this purpose, we compare DepPattern with MaltParser[5] (Nivre et al., 2007b), one of the top performing systems in the CoNLL shared tasks on multilingual dependency parsing in 2006 and 2007 (Hall and Nilsson, 2006; Nivre et al., 2007a). Experiments were performed on two languages: Portuguese and Spanish. The reason of making experiments on only two languages is the very high cost required to adapt the outputs of our system to the test data set.

5.1 Training and Test Corpora

The experiments were performed by making use of two dependency treebanks: Spanish Ancora 2.0 (Recasens and Martí, 2010) and Portuguese Bosque 8.0 (Afonso et al., 2002), also used by participants at the CoNLL 2006 shared task. In order to have a similar experimental setup for the two languages, each corpus was divided in a training part containing $115,000$ words and a test set with $100,000$ words. The two training corpora were only used to build the statiscal model of MaltParser. DepPattern, which is a grammar-based system, does not require any training corpus.

5.1.1 MaltParser's Optimization

We used MaltParser 1.7.1, equipped with nine different transition-based parsing algorithms. To select the best algorithm running in linear time (5 out 9: Nivre eager, Nivre standard, Stack proj, planar, and 2-planar), we validated the target algorithms on the Portuguese test data set. The best performance was achieved by 2planar (Gómez-Rodríguez and Nivre, 2010), based on arc-eager algorithm, even if the difference among the five tested algorithms was not statistically significant. So, the 2planar strategy was chosen to be compared with DepPattern.

All experiments were performed using a classifier based on support vector machines, as implemented in the LIBSVM package (Chang and Lin, 2001). Finally, we have reused the PoS tags and feature representation that produced the best results on Portuguese and Spanish for the MaltParser

[5]`htpp://www.maltparser.org/`

system at CoNLL shared task.

5.1.2 Test data sets

The two main data sets are the following:

bosque-test More than 3,000 sentences with 100M tokens taken from the Portuguese treebank.

ancora-test More than 3,000 sentences with 100M tokens taken from the Spanish treebank.

In order to compare MaltParser with DepPattern, we must address three problems when preparing the test data set:

- The tagsets and feature structures required by DepPattern are not the same as those available in the treebanks to train MaltParser.
- Each treebank was annotated following different criteria regarding some linguistic phenomena. Besides, there are also differences of criteria between DepPattern and some dependency solutions found in the treebanks.
- MaltParser can take advantage of the fact that both training and test sets belong to the same domain.

DepPattern requires as input PoS tagged text with the format provided by Tree-Tagger or FreeLing. Ancora 2.0 was annotated with FreeLing tags, but it is not the case of Bosque 8.0. In order to permit DepPattern to have the same entry as MaltParser for the Portuguese corpus, the PoS tags and features of Bosque 8.0 were converted into FreeLing-based PoS tags. Note that this PoS tag conversion allows us to apply MaltParser, in combination with FreeLing, to raw text. First, the PoS tags and feature representation of the two training sets were transformed into the FreeLing format. Then, the best MaltParser configuration was trained on these two sets and, finally, it was applied on text previously tagged with FreeLing. The results obtained were very similar (even slightly better) to those obtained with the PoS tags and features of the original treebanks. Conversion scripts are freely available.[6]

However, the main problem to evaluate DepPattern and to compare it with other systems is to minimize the noise derived from the choice of different linguistic criteria. In particular, we found in DepPattern grammars several linguistic decisions to define dependencies that are very different from those found in the exploited treebanks. For instance:

- All clause arguments (including the subject) are dependent on lexicalized verbs, and not on light or auxiliary verbs.
- All members of a coordination are dependent on the conjunction and not on the first coordinated member.

DepPattern grammars follow these two criteria. Bosque 8.0 follows neither of the two, and Ancora 2.0 just follows the first one. So, there are three different dependency criteria for dealing with these two linguistic phenomena, which have large coverage. In addition, there are many other conflicts to be considered: DepPattern analyzes the Portuguese possesive expressions in a different way as the treebank annotators: in 'a sua mulher' (*his wife*), 'sua' is dependent of 'mulher' and 'a' of 'sua'. However, in the Portuguese treebank, 'a' and 'sua' both depend of 'mulher'. There are also different decisions for the internal dependencies of periphrastic verbal expressions: e.g. 'tiene que ir' (*have to go*). Particle 'que' (*to*) can be either dependent of 'tiene' or of 'ir'. All these differences are a serious drawback for comparing our grammar-based parser against a corpus-driven system. To solve the problem, we adapted our generic Romance Grammar to each treebank. As a result, we generated both a Portuguese parser adapted to Bosque and a Spanish parser adapted to Ancora. Another solution would have been to create conversor scripts to automatically modify the treebanks. However, as the algorithms required are not trivial, in particular for complex phenomena such as coordination, this strategy was not considered.

The third problem is related to the content similarity between the training and the test corpora, which could benefit the data-driven system. Apart from the two data sets extracted from the treebanks, we also built a small gold standard, called *open-test*, by manually analyzing sentences extracted from different sources. The description of this new test data set is the following:

open-test 42 Portuguese sentences with about 1K words taken from different sources with a

[6] `http://gramatica.usc.es/~gamallo/resources/depcorpus-test.tgz`

	Precision	Recall	F-score
ancora-test (es)	84.6	79.7	82
bosque-test (pt)	84.1	79.2	81.6
open-test (pt)	86.2	76.6	81.1

Table 1: Evaluation of DepPattern (unlabeled dependencies)

	Precision	Recall	F-score
ancora-test (es)	85	85	**85**
bosque-test (pt)	88.2	88.2	**88.2**
open-test (pt)	81.3	81.3	**81.3**

Table 2: Evaluation of MaltParser (unlabeled dependencies)

variety of genres and domains: Wikipedia (scientific domain and encyclopedic genre), a Portuguese novel by Machado Assis (literature genre), documents of the European Commission (economy domain).

Both *bosque-test* and *ancora-test* consist of sentences belonging to the same journalist genre as those found in the training corpus. By contrast, *open-test* consists of miscellaneous sentences whose content is distributed through different genres and domains. So it can be considered as an open-content test set. The dataset is freely available[7]

5.2 Evaluation

To evaluate the performance of the two systems, it is necessary to take into account that DepPattern produces partial parses. This system assigns the dependency relation '0' (or Root) to all unattached tokens. So, it is relevant to make use of precision and recall, instead of accuracy, to measure the performance of the DepPattern parsers. For MaltParser, precision and recall are the same: these values correspond to the unlabeled attachment score (UAS).

Furthermore, as the dependency labels of DepPattern are very different from those found in the two treebanks, the evaluation was restricted to unlabeled dependencies. Parser evaluation is conducted by comparing the dependencies guessed by the system with manually revised dependencies. Given these two data sets, precision and recall are defined as in (Lin, 1998). As '0' values, associated to unattached tokens, are considered as not found dependencies, they are relevant to measure recall in DepPattern. Punctuation marks are ignored in our evaluation.

Tables 1 and 2 show the results obtained with DepPattern and MaltParser, respectively, when applied on the different testing sets. The results obtained by MaltParser represent in fact the *UAS* of the system, since precision and recall are the same. MaltParser outperforms DepPattern on both *bosque-test* and *ancora-test*.

The scores we obtained using MaltParser follow the same tendency (even if they are not identical) of those obtained at the CoNLL 2006 shared task, where the system achieved 91% accuracy on Portuguese and 85 on Spanish (for unlabeled attachment scores). In our experiments, MaltParser obtained 88.2% and 85, respectively. The differences between the Portuguese scores at CoNLL and those obtained in our experiment could derive from small changes in the optimization procedure, and from the size of the training corpus. Notice that the performance of MaltParser is quite different across our three data sets: 88.2% (bosque) 85 (ancora), and 81.3 (open). By contrast, DepPattern achieves similar results in all the tests.

In the content-open test, we observe that whereas MaltParser gets down from 88.2 accuracy to 81.3, DepPattern keeps a similar score in the three datasets. It seems that the change of domain affects the performance of the data-driven system. By contrast, the grammar-based parser keeps a similar performance across domains and genres. It seems to be also most stable across different languages: it achieves similar results in Spanish and Portuguese because, on the one hand, these languages are very close and, on the other, DepPattern only use those grammar rules shared by the two languages. However, it is not easy to explain why MaltParser behaves in a very different way on two languages which are very similar in terms of grammar.

Finally, we also verified whether the two systems are complementary. This was made by measuring the statistical correlation between the results obtained by the two types of parsers. For this purpose, we analysed the Pearson correlation between the parses resulting from both DepPatter and MaltParser, in order to verify if they tend to make the same correct decisions. The Pearson coefficient obtained was low, namely 0.14, with only 69% of shared correct decisions. This

[7] `http://gramatica.usc.es/~gamallo/resources/depcorpus-test.tgz`

means that the two systems are complementary since many of the correct dependencies they guess are not the same. In sum, they are good in different ways. This insight essentially encourages to the pursuit of hybrid approaches and parser combinations, since both strategies seem to be complementary and should work together to produce more efficient results.

6 Conclusions

The work described in the article can be seen as a contribution to improve old parsing strategies introduced at the end of the twentieth century, when most efficient techniques were based on rules/FST and constructive approaches. In particular, we described a grammar-driven parser based on FST, called compression parsing, which takes into account some elements from deterministic and incremental dependency parsing, namely *Arc* and *Reduce* transitions. This compression method, implemented in DepPattern, was compared with a data-driven, transition-based system: MaltParser.

The cost and effort of developing compression parsers for several languages is not very high, since they can achieve reasonable performance using just very simple, multilingual, and general-purpose grammars. In this article, we have also introduced a simple methodology to write multilingual and general-purpose grammars.

In future work, it would be interesting to compare a variety of grammar-driven systems by measuring, not only their performance, but also the complexity of the underlying grammar: number of rules, size (in bytes) of the source files, etc. It should also be important to quantify and compare the cost and effort of both writing grammars and building treebanks. Moreover, to complement quantitative evaluation, it will be necessary to define objective protocols to compare parsers on the basis on qualitative evaluation (Lloberes et al., 2014).

Finally, we claim that FST-based parsing techniques simulate how we solve problems quickly, by taking first easy decisions which, in turn, make it easier to solve further complex tasks. However, these parsing techniques are far from simulating two other interesting cognitive operations: first, how grammars are learnt and, second, how sentences are processed. Data-driven approaches can be seen as a good approximation to the way humans learn grammars, while incremental left-to-right parsing can be seen as a simulation of how humans process and understand input sentences. Here, a question arises: would it be possible to define a method taking advantage of all those parsing strategies? In other words, could be it possible to model a strategy that learns grammar rules from data, orders them as cascades of progressively more complex transducers, and applies them to sentences in an incremental way? A method provided with these three 'human properties' would be closer to the canonical systems in Artificial Intelligence, since the main objective would be, not only to produce parse trees, but also to simulate how humans understand sentences.

Acknowledgement

This work has been supported by RedPLIR, Xunta de Galicia, ref: CN2014/034.

References

Steven Abney. 1996. Partial parsing via finite-state cascades. In *Proceedings of the ESSLLI'96 Robust Parsing Workshop*, pages 8–15.

Susana Afonso, Eckhard Bick, Renato Haber, and Diana Santos. 2002. Floresta sintá(c)tica": a treebank for portuguese. In *LREC 2002, the Third International Conference on Language Resources and Evaluation*, pages 1698–1703, Las Palmas de Gran Canaria, Spain.

Salah Ait-Mokhtar and Jean-Pierre Chanod. 1997. Incremental finite-state parsing. In *Proceedings of the 5th Conference on Applied Natural Language Processing (ANLP-97)*, Washington, DC, USA.

Salah Ait-Mokhtar, Jean-Pierre Chanod, and Claude Roux. 2002. Robustness beyond shallowness: Incremental deep parsing. *Natural Language Engineering*, 8(2/3):121–144.

Xavier Carreras. 2007. Experiments with a higher-order projective dependency parser. In *EMNLP/CoNLL*.

Chih Chang and Chih J. Lin, 2001. *LIBSVM: a library for support vector machines.*

Fabio Ciravegna and Alberto Lavelli. 2002. Full parsing approximation for information extraction via finite-state cascades. *Natural Language Engineering*, 8(2/3):145–165.

Hal Daumé-III and Daniel Marcu. 2007. Frustratingly easy domain adaptation. In *45th Annual Meeting of the Association for Computational Linguistics (ACL)*, Prague, Czech Republic.

Hal Daumé-III. 2006. Domain adaptation for statistical classifiers. *Artificial Intelliegence Research*, 26:101–126.

Jason Eisner and Noah A. Smith. 2010. Favor short dependencies: Parsing with soft and hard constraints on dependency length. In Harry Bunt, Paola Merlo, and Joakim Nivre, editors, *Trends in Parsing Technology: Dependency Parsing, Domain Adaptation, and Deep Parsing*, chapter 8, pages 121–150. Springer.

Pablo Gamallo and Isaac González. 2011. A grammatical formalism based on patterns of part-of-speech tags. *International Journal of Corpus Linguistics*, 16(1):45–71.

Pablo Gamallo and José Ramom Pichel. 2008. Learning Spanish-Galician Translation Equivalents Using a Comparable Corpus and a Bilingual Dictionary. *LNCS*, 4919:413–423.

Pablo Gamallo, Marcos Garcia, and Santiago Fernández-Lanza. 2012. Dependency-based open information extraction. In *ROBUS-UNSUP 2012: Joint Workshop on Unsupervised and Semi-Supervised Learning in NLP at the 13th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2012)*, Avignon, France.

Marcos Garcia and Pablo Gamallo. 2010. Using morphosyntactic post-processing to improve postagging accuracy. In *PROPOR-2010*, Porto-Alegre, Brasil.

Marcos Garcia and Pablo Gamallo. 2011. Dependency-based text compression for semantic relation extraction. In *Workshop on Information Extraction and Knowledge Acquisition (IEKA 2011) at 8th International Conference on Recent Advances in Natural Language Processing (RANLP 2011)*, pages 21–28.

Daniel Gildea. 2001. Corpus variation and parser performance. In *2001 EMNLP Conference*, Pittsburgh, PA.

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 Association for Computational Linguistics*, pages 742–750, Stroudsburg, PA, USA.

Carlos Gómez-Rodríguez and Daniel Fernández-González. 2012. Dependency parsing with undirected graphs. In *13th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 66–76, Avignon, France.

Carlos Gómez-Rodríguez and Joakim Nivre. 2010. A transition-based parser for 2-planar dependency structures. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, pages 1492–1501, Stroudsburg, PA, USA.

Carlos Gómez-Rodríguez, D. Fernández-González, and V. Bilbao. 2014. Undirected dependency parsing. *Computational Intelligence*.

Gregory Grefenstette. 1996. Light parsing as finite-state filtering. In *Workshop on Extended Finite State Models of Language ECAI'96*, Budapest, Hungary.

Johan Hall and Jens Nilsson. 2006. CoNLL-X shared task on multilingual dependency parsing. In *The tenth CoNLL*.

Aravind Joshi. 1996. A parser from antiquity: An early application of finite state transducers to natural language parsing. In *ECAI-96 Workshop on Extendend Finite State Models of Languages*, Budapest, Hungary.

Dimitrios Kokkinakis and Sofie Johansson Kokkinakis. 1999. A cascaded finite-state parser for syntactic analysis of swedish. In *The 9th EACL*, Bergen, Norway.

Dekang Lin. 1998. Dependency-Based Evaluation of MINIPAR. In *Workshop on Evaluation of Parsing Systems*, Granada, Spain.

Marina Lloberes, Irene Castellón, Lluís Padró, and Edgar González. 2014. Partes. test suite for parsing evaluation. *Procesamiento del Lenguaje Natural*, 53:87–94.

André F. T. Martins, Noah A. Smith, Eric P. Xing, Pedro M. Q. Aguiar, and Mário A. T. Figueiredo. 2010. Turboparsers: Dependency parsing by approximate variational inference. In *Empirical Methods in Natural Language Processing (EMNLP'10)*, Boston, USA.

Ryan McDonald and Fernando Pereira. 2006. Online learning of approximate dependency parsing algorithms. In *EACL-2006*, pages 81–88.

Joaquim Nivre, Johan Hall, and Jens Nilson. 2004. Memory-based dependency parsing. In *Proceedings of CoNLL*, pages 49–56.

Joakim Nivre, Johan Hall, Sandra Kübler, Ryan McDonald, Jens Nilson, Sebastian Riedel, and Deniz Yuret. 2007a. The CoNLL-2007 shared task on dependency parsing. In *Proceedings of the Shared Task Session of EMNLP-CoNLL 2007*, pages 915–932, Prague, Czech Republic.

Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Gülsen Eryigit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. 2007b. Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(2):115–135.

Joaquim Nivre. 2004. Incrementality in deterministic dependency parsing. In *ACL Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*, pages 50–57.

Joaquim Nivre. 2005. Dependency grammar and dependency parsing. Technical Report MSI report 05133, Växjö University: School of Mathematics and Systems Engineering.

Kemal Oflazer. 2003. Dependency parsing with an extended finite-state approach. *Computational Linguistics*, 29(4):515–544.

Lluís. Padró and Evgeny Stanilovsky. 2012. Freeling 3.0: Towards wider multilinguality. In *Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey.

Marta Recasens and M. Antònia Martí. 2010. AnCora-CO: Coreferentially annotated corpora for spanish and catalan. *Language Resources and Evaluation*, 315-345(4):315–345.

Laura Rimell, Stephen Clark, and Mark Steedman. 2009. Unbounded dependency recovery for parser evaluation. In *Conference on Empirical Methods in Natural Language Processing*, pages 813–821, Singapore.

Helmut Schimd. 1995. Improvements in part-of-speech tagging with an application to german. In *ACL SIGDAT Workshop*, Dublin, Ireland.

Stephen Tratz and Eduard Hovy. 2011. A fast, effective, non-projective, semantically-enriched parser. In *In Proceedings of EMNLP. Joseph Turian, Lev-Arie Ratinov, and Yoshua Bengio*.

Yannick Versley. 2014. Experiments with easy-first nonprojective constituent parsing. In *Proceedings of the First Joint Workshop on Statistical Parsing of Morphologically Rich Languages and Syntactic Analysis of Non-Canonical Languages*, pages 39–53, Dublin, Ireland.

Hiroyasu Yamada and Yuji Matsumoto. 2003. Statistically dependency analysis with support vector machines. In *Proceedings of 8th International Workshop on Parsing Technologies (IWPT)*, pages 195–206.

Stacking or Supertagging for Dependency Parsing – What's the Difference?

Agnieszka Faleńska[§,◇], **Anders Björkelund**[§], **Özlem Çetinoğlu**[§] and **Wolfgang Seeker**[§]

[§]Institute for Natural Language Processing, University of Stuttgart, Germany
[◇]Institute of Computer Science, University of Wrocław, Poland
{falensaa,anders,ozlem,seeker}@ims.uni-stuttgart.de

Abstract

Supertagging was recently proposed to provide syntactic features for statistical dependency parsing, contrary to its traditional use as a disambiguation step. We conduct a broad range of controlled experiments to compare this specific application of supertagging with another method for providing syntactic features, namely stacking. We find that in this context supertagging is a form of stacking. We furthermore show that (i) a fast parser and a sequence labeler are equally beneficial in supertagging, (ii) supertagging/stacking improve parsing also in a cross-domain setting, and (iii) there are small gains when combining supertagging and stacking, but only if both methods use different tools. The important consideration is therefore not the method but rather the diversity of the tools involved.

1 Introduction

We present a systematic comparison of two methods that have been proposed to improve statistical dependency parsers: *supertagging* and *stacking*.

Supertags are labels for tokens much like POS tags but they also encode syntactic information, e.g. the head direction or the subcategorization frame. Supertagging was developed for deep grammar formalisms as the disambiguation of supertag assignment *prior to parsing* (Bangalore and Joshi, 1999; Clark and Curran, 2004; Ninomiya et al., 2006). Recently, it was presented as a method to provide syntactic information to the feature model of a statistical dependency parser. Ambati et al. (2013; 2014) provide CCG supertags to a dependency parser, whereas Ouchi et al. (2014) extract their supertag tag set from a dependency treebank (see Figure 1). In this paper, we adopt this *particular definition* and take supertagging as a way of incorporating syntactic features instead of the traditional use for disambiguation.

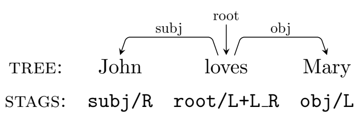


Figure 1: Supertags that are derived from dependency trees for each token. They encode the label, the head direction, and the presence of left and right dependents.

Parser stacking was introduced by Nivre and McDonald (2008) and Martins et al. (2008). In stacking, two parsers are run in sequence so that the second parser can use the output of the first parser as features, for example, whether a particular arc was already predicted by the first parser.

When supertags were first proposed by Joshi and Bangalore (1994), they called supertagging *almost parsing*, because supertags anticipate a lot of syntactic disambiguation. In stacking, the first step is running a parser, or in other words: *real parsing*. In this paper, we investigate the difference between *almost* and *real parsing* for improving a statistical dependency parser.

We conduct an extensive number of comparative experiments with two state-of-the-art dependency parsers and a state-of-the-art sequence labeler on 10 different data sets. In the first set of experiments, we use only the two parsers and compare both methods in artificial and realistic settings. In the second set of experiments, we control for the methods and compare different ways of realizing them. In the last set, we evaluate the benefit of combining both methods.

Intuitively, stacking should give higher improvements than the version of supertagging defined by Ouchi et al. (2014), since trees in stacking are more informative than supertag sequences in supertagging. However, our experiments show that both methods perform on par. Based on an in-depth analysis of these findings, we argue that supertagging is a form of stacking.

One apparent advantage of supertagging is the fact that one can predict supertags without a parser and thus possibly faster. However, greedy transition-based parsers are extremely fast as well. We show that the output of a CRF sequence labeler and a greedy transition-based parser are of equal usefulness when used in supertagging. This setup suggests application to large-scale (e.g. web) data. We test both methods on the English Web Treebank (Bies et al., 2012) and show that they also improve parsing in a cross-domain setting.

Our experiments on combining supertagging and stacking show small gains only when supertags and trees are predicted by different tools. Surdeanu and Manning (2010) demonstrate that diversity of algorithms is important when stacking parsers. Since supertagging is a form of stacking, this also holds for supertagging, and we argue that this is a more important factor than the choice between the two methods.

We give background on supertagging and stacking in Section 2 and describe our experimental setup in Section 3. We present our experiments in Sections 4 to 6 and conclude with Section 7.

2 Background

The term supertag originated in Joshi and Bangalore (1994) as an elementary structure associated with a lexical item. These elementary structures carry more information than POS tags, hence the name super POS tags or supertags. Within Lexicalized Tree Adjoining Grammar (LTAG) (Schabes et al., 1988) supertags correspond to trees that localize dependencies. A supertagger assigns supertags to each word of a sentence. A parser then combines these structures into a full parse (Bangalore and Joshi, 1999) that leads to simplified and faster parsing. The same approach applied to Combinatory Categorial Grammar (CCG) (Clark and Curran, 2004) and Head-Driven Phrase Structure Grammar (HPSG) (Ninomiya et al., 2006) speeds up the parser dramatically.

Foth et al. (2006) were the first to utilize supertags in a dependency parsing context by incorporating them as soft constraints into their rule-based parser (Foth et al., 2004). In LTAG, CCG, or HPSG supertags are the elementary components of the framework in question. In Foth et al. (2006), supertags are specifically designed to capture syntactic properties.

Ouchi et al. (2014) use supertags as features in a statistical dependency parser for English. Ambati et al. (2014) instead utilize CCG categories for English and Hindi. Both demonstrate significant improvements. Björkelund et al. (2014) extend the positive results to nine other languages.

Another way of exploiting one parser's output as features in another parser is stacking. Nivre and McDonald (2008) define a simple set of local features that mark whether an arc is present in the input tree. They demonstrate that stacking parsers leads to higher parsing accuracy than a non-stacked baseline. Martins et al. (2008) extend this feature set to include non-local information, e.g. information about siblings and grandparents of dependents. However, the additional non-local features provide only minor further gains over the local ones if the parser itself already uses non-local features.

Surdeanu and Manning (2010) present a study on parser stacking for English. They find that one important factor is the diversity of the parsing algorithms involved. Specifically, stacking a parser on itself does not lead to gains. This effect was also observed by Martins et al. (2008).

3 Experimental Setup

3.1 Data Sets and Preprocessing

We perform experiments on the data from the SPMRL 2014 Shared Task (Seddah et al., 2014), which consists of data sets for 9 languages (see Table 1). To these 9, we add the English Penn Treebank converted to Stanford Dependencies.[1] We use sections 2-21 for training, 24 as development set and 23 as test set.

Contrary to most previous work, we use automatically predicted preprocessing in all the parsing experiments. POS tags and morphological features are jointly assigned using MarMoT[2] (Müller et al., 2013), a state-of-the-art morphological CRF tagger. To improve tagging accuracy we integrate the analyses of language-specific morphological analyzers as additional features into MarMoT (see Table 1). We use the mate-tools[3] for lemmatiza-

[1]We use version 3.4.1 of the Stanford Parser from `http://nlp.stanford.edu/software/lex-parser.shtml`

[2]`https://code.google.com/p/cistern/`

[3]`https://code.google.com/p/mate-tools/`

tion. We annotate the training sets via 5-fold jackknifing.

ISO	Language	Morphological Analyzer
ar	Arabic	AraMorph, a re-impl. of Buckwalter (2002)
eu	Basque	Apertium (Forcada et al., 2011)
fr	French	An extension of Zhou (2007)
he	Hebrew	Analyzer from Goldberg and Elhadad (2013)
de	German	SMOR (Schmid et al., 2004)
hu	Hungarian	Magyarlanc (Zsibrita et al., 2013)
ko	Korean	HanNanum (Park et al., 2010)
pl	Polish	Morfeusz (Woliński, 2006)
sv	Swedish	Granska (Domeij et al., 2000)

Table 1: Analyzers used in the tagger.

3.2 Supertag Design

Foth et al. (2006) experiment with different tag set designs and show that richer supertags improve their parser's accuracy more. However, richer tags increase the tag set size considerably and make it more difficult to predict them automatically.

Ouchi et al. (2014) test two models for English. Model 1 includes the relative head position of a word (**hdir**), its dependency relation (**label**), and information about dependents to the left or right (**hasLdep, hasRdep**). The tag set is derived from the treebank, an example is shown in Figure 1. Model 2 additionally uses dependency relations of obligatory dependents of verbs. The difference between the two models has no impact on the performance of a parser, however. Björkelund et al. (2014) find the same effect for the same models on nine other languages.

ar	eu	fr	de	he	hu	ko	pl	sv	en
42	179	128	239	196	280	74	113	253	222

Table 2: Tag set sizes for training sets.

Based on these results we decided to use Model 1 in all of the experiments. The supertags are extracted from the respective training sets and follow the template **label/hdir+hasLdep_hasRdep**. Table 2 gives the tag set sizes for each data set.

3.3 Notation

We denote stacking and supertagging by STACK and STAG, respectively. When a tool *y* uses the output of another tool *x*, we mark this by superscript and subscript. For example, STACK_x^y means that tool *y* uses the output of tool *x* in stacking. Similarly, STAG_x^y means that tool *y* uses the supertags predicted by tool *x*. We follow Martins et al. (2008)'s terminology and call *x* the Level 0 tool and *y* the Level 1 tool.

3.4 Parsers and Feature Models

In the experiments, Level 1 tools will always be dependency parsers since we are interested in the effect of supertagging and stacking on parsing performance. We experiment with one graph-based and one transition-based parser to cover the two major paradigms in dependency parsing.

We extend the parsers' baseline feature sets in two directions: (1) to extract features for stacking, i.e., to extract features from a provided dependency tree, and (2) to extract features from a sequence of supertags. For stacking, features are taken from Nivre and McDonald (2008) and slightly adapted to our setting. For supertagging, we mirror the features from stacking to the best extent possible given the more limited information that is contained in the supertags to begin with.

We note that feature engineering can be done more elaborately both for stacking (Martins et al., 2008) and supertagging (Ouchi et al., 2014). However, since not all types of features that can be extracted necessarily carry over from one method to the other, a simpler feature set is more useful for a comparison. Moreover, both of the aforementioned papers only demonstrate minor performance gains with more elaborate features.

Transition-based Parser. We use the parser by Björkelund and Nivre (2015) as our transition-based parser. It uses the arc-standard decoding algorithm extended with a SWAP transition (Nivre, 2009) to handle non-projective structures.[4] The system applies arc transitions between the two topmost items of the stack (denoted s_0 and s_1). The lazy SWAP oracle by Nivre et al. (2009) is used during training. The parser is globally trained using beam-search and early update (Zhang and Clark, 2008). The implementation uses the passive-aggressive perceptron (Crammer et al., 2006) and a hash kernel for feature mapping following Bohnet (2010). The parser is trained for 25 iterations using a beam size of 20. We omit the definition of the baseline feature set of the transition-based parser, however it is primarily based on that of Zhang and Nivre (2011) with adaptations to the arc-standard setting.

Table 3 outlines the feature templates used for stacking and supertagging. The predicates

[4]This parser is available on the second author's website.

Stacking features	
head$_G(s_0) = s_1$	head$_G(s_1) = s_0$
hdir$_G(s_0)$	hdir$_G(s_1)$
label$_G(s_0)$	label$_G(s_1)$
hasL$_G(s_0) \oplus$ pos(ldep(s_0))	hasR$_G(s_0) \oplus$ pos(rdep(s_0))
hasL$_G(s_1) \oplus$ pos(ldep(s_1))	hasR$_G(s_1) \oplus$ pos(rdep(s_1))
Supertag features	
stag$_S(s_0)$	stag$_S(s_1)$
label$_S(s_0)$	label$_S(s_1)$
hdir$_S(s_0)$	hdir$_S(s_1)$
hasL$_S(s_0)$	hasR$_S(s_1)$
stag$_S(s_0) \oplus$ pos(ldep(s_0))	stag$_S(s_0) \oplus$ pos(rdep(s_0))
stag$_S(s_1) \oplus$ pos(ldep(s_1))	stag$_S(s_1) \oplus$ pos(rdep(s_1))
hasL$_S(s_0) \oplus$ hdir$_S(s_1)$	hasR$_S(s_1) \oplus$ hdir$_S(s_0)$
hasL$_S(s_0) \oplus$ pos(ldep(s_0))	hasR$_S(s_1) \oplus$ pos(rdep(s_1))
hasR$_S(s_0) \oplus$ pos(rdep(s_0))	hasL$_S(s_1) \oplus$ pos(ldep(s_1))

Table 3: Feature templates used for stacking and supertagging in the transition-based parser. $\oplus$ denotes conjunctions of basic templates. All templates are conjoined with the POS tag of the topmost stack items s_0 and s_1.

head$_X(d)$, hdir$_X(d)$, label$_X(d)$, hasL/R$_X(d)$, and stag$_X(d)$ extract the head of d, the direction of d's head, the arc label of d, whether d has left/right dependents, and the supertag according to the Level 0 prediction X. X is either a dependency graph (in stacking) or a supertag assignment (in supertagging), denoted G and S in Table 3, respectively. The predicates l/rdep(d) extract the leftmost/rightmost dependent of d given the current parser state. pos(d) extracts the POS tag of d, with a special placeholder if d is undefined.

The stacking features are mostly taken from Nivre and McDonald (2008) with the exception of the last two rows. These features encode whether d should have left/right dependents according to G conjoined with whether d has left/right dependents in the current configuration. We added these features because existence of left/right dependents is also encoded in the supertags. Conjoining the existence of left/right dependents according to the Level 0 predictions with the POS tag of left/right dependents in the current parser state thus encodes whether dependents were attached or not. Since the arc-standard algorithm works bottom-up, every token needs to collect all its dependents before it can be attached to its own head.

The supertag features mimic the information provided by stacking. For instance, in stacking the Level 0 predictions explicitly include whether s_0 is the head of s_1. In supertagging this is approximated by combining the direction of the head of s_1 with whether s_0 expects dependents on the left.

Stacking features	
head$_G(d) = h$	
label$_G(d)$	
head$_G(d) = h \oplus$ label$_G(d)$	
Supertag features	
stag$_S(h)$	stag$_S(d)$
label$_S(d)$	hdir$_S(d)$
label$_S(d) \oplus$ hdir$_S(d)$	
hasL$_S(h)$	hasR$_S(h)$
hdir$_S(d) \oplus$ hasL$_S(h)$	hdir$_S(d) \oplus$ hasR$_S(h)$
label$_S(d) \oplus$ hasL$_S(h)$	label$_S(d) \oplus$ hasR$_S(h)$
label$_S(d) \oplus$ hdir$_S(d) \oplus$ hasL$_S(h)$	
label$_S(d) \oplus$ hdir$_S(d) \oplus$ hasR$_S(h)$	

Table 4: Feature templates used for stacking and supertagging in the graph-based parser. $\oplus$ denotes conjunctions of basic templates. All templates are conjoined with the direction of that arc and with the POS tag of the head and the dependent.

Graph-based Parser. The graph-based parser we use is TurboParser,[5] which solves the parsing task by doing global inference using a dual decomposition algorithm and outputs non-projective structures natively (Martins et al., 2013).

Table 4 shows the stacking and supertagging features as we implemented them in TurboParser. They are synchronized with the features for the transition-based parser where possible. We extract these features only on first-order factors, with d and h denoting the dependent and the head, respectively. Unlike in Nivre and McDonald (2008), the features cannot access the label of the current arc during feature extraction, as it is automatically combined with the features after the extraction.

Like in the transition-based parser, supertag and stacking features are modeled to capture similar information. However, features that combine information about dependents of dependents with information about the head are not included since these would require higher-order factors.

3.5 Evaluation

We evaluate the parsing experiments using *Labeled Attachment Score* (LAS).[6] We mark statistical significance against respective baselines by † and ‡, denoting p-value < 0.05 and p-value < 0.01 respectively. Significance testing is carried out using the Wilcoxon signed-rank test. Averages and oracle experiments are not tested for significance.

[5]We use version 2.0.1 from `http://www.ark.cs.cmu.edu/TurboParser/`. We train TurboParser with MODEL-TYPE=FULL which uses third-order features.

[6]The ratio of tokens with a correct head and label to the total number of tokens in the test data.

		avg.	ar	eu	fr	de	he	hu	ko	pl	sv	en
①	BL[GB]	84.45	84.99	82.79	83.95	88.53	79.41	83.98	86.18	84.96	79.59	90.12
②	$\text{STAG}^{\text{GB}}_{\text{TB}}$	85.15	85.54‡	83.24‡	84.33‡	89.14‡	80.10†	85.52‡	86.48	85.48	80.63‡	90.99‡
③	$\text{STACK}^{\text{GB}}_{\text{TB}}$	85.16	85.65‡	83.32‡	84.29‡	89.15‡	80.03‡	85.46‡	86.59‡	85.52†	80.66‡	90.95‡
④	BL^{TB}	84.37	85.09	81.77	83.47	87.89	79.70	85.25	85.71	84.34	79.97	90.54
⑤	$\text{STAG}^{\text{TB}}_{\text{GB}}$	85.01	85.58‡	82.99‡	83.88‡	88.88‡	80.04	85.37	86.31‡	85.03	81.04‡	90.98‡
⑥	$\text{STACK}^{\text{TB}}_{\text{GB}}$	85.08	85.60‡	83.14‡	84.16‡	88.91‡	80.30	85.38	86.06†	85.06†	81.32‡	90.88‡

Table 5: Parsing results (LAS) on test sets.

4 Comparing Supertagging and Stacking

The purpose of the following experiments is to compare supertagging and stacking and to derive some conclusions about their relationship to each other. We use one parser as the Level 0 parser and the other one as Level 1 parser. In stacking, the Level 1 parser exploits the tree produced by the Level 0 parser as additional features. In supertagging, we derive the supertag of each token from the tree that is output by the Level 0 parser. The Level 1 parser then uses these supertags as additional features. Although supertags are normally predicted with sequence labelers, using a parser on Level 0 in both cases ensures that the only difference between the two settings is the means by which the information is given to the Level 1 parser, i.e. as a tree or as a sequence of supertags. Figure 2 illustrates this setup.

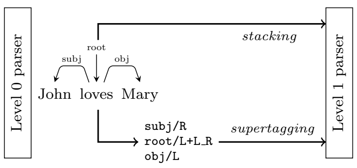


Figure 2: Setup for comparing stacking and supertagging.

The training sets are annotated with predicted dependency trees or supertags via 5-fold jackknifing. In the tables, **GB** stands for the graph-based parser and **TB** for the transition-based parser.

4.1 Supertagging and Stacking Accuracy

First of all we convince ourselves that both strategies, supertagging and stacking, indeed improve over the baseline. Table 5 gives the performance of the Level 1 parser on the test sets: In the baseline setting (**BL**) the parser is run without any additional information. STAG and STACK show the performance of the Level 1 parser when provided with supertags or a tree from the Level 0 parser.

As demonstrated by previous work, both stacking and supertagging consistently improve the parsing performance of the Level 1 parser. Moreover, both methods improve the parsing accuracies to the same extent, with the average improvements about 0.7% points absolute for both the graph-based and the transition-based parser. Almost all of the improvements are statistically significant, with a few exceptions, most notably Polish. For supertagging, our results confirm the findings by Ouchi et al. (2014) and Ambati et al. (2014). The stacking results are in line with Nivre and McDonald (2008) and Martins et al. (2008). Here, it is worth noting that even though dependency parsers have markedly advanced since 2008, the fact remains that stacking parsers improves performance.

We now continue with a more in-depth analysis to find out where the improvements are coming from. We perform the analysis on the development sets in order to not compromise our test sets. The corresponding accuracies for the development sets can be found in Tables 6 and 7 in rows ① to ③.

4.2 In-Depth Analysis

The overall improvements with supertagging and stacking are similar, but they might still come about in different ways. To investigate this, we follow McDonald and Nivre (2007) and look into accuracy distributions of comparable systems relative to sentence length and dependency length, i.e. the distance between the dependent and the head.

We present the analysis on the concatenation of all the development sets. We also looked at the corresponding plots for the individual treebanks. While the absolute numbers vary across the different data sets, the relative differences between the baseline, supertagging, and stacking models are consistent with the concatenation.

Figure 3 gives the accuracy of both parsers relative to sentence length in bins of size 10. Bin sizes are represented as grey bars.[7]

[7]Note that if there are fewer items in a bin, the curves are more sensitive to small absolute changes.

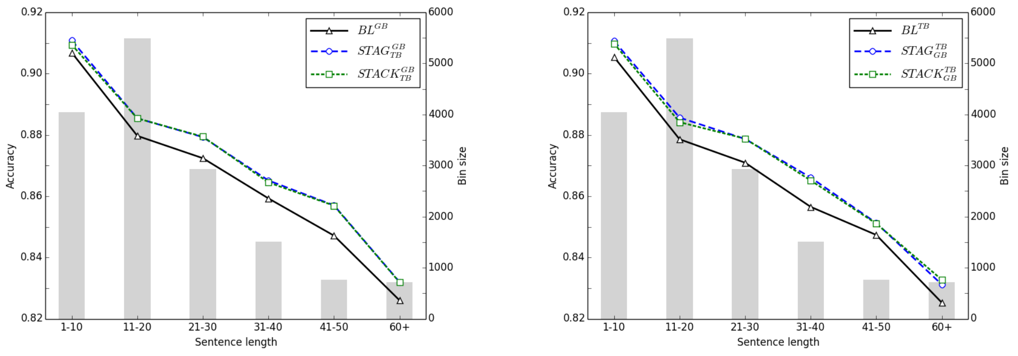


Figure 3: The accuracy of the graph-based (left) and transition-based (right) parser relative to sentence length.

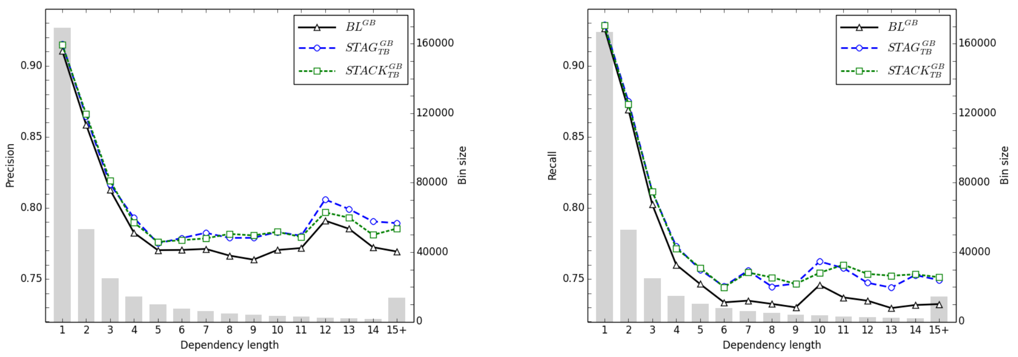


Figure 4: The dependency precision and recall of the graph-based parser relative to dependency length.

		avg.	ar	eu	fr	de	he	hu	ko	pl	sv	en
①	BL^{GB}	84.16	85.64	83.37	84.78	91.46	78.71	82.60	86.08	85.17	75.24	88.59
②	$\text{STAG}^{\text{GB}}_{\text{TB}}$	84.81	86.25‡	83.96‡	85.05†	92.10‡	79.36†	83.92‡	86.24	85.52†	76.27†	89.47‡
③	$\text{STACK}^{\text{GB}}_{\text{TB}}$	84.79	86.25‡	83.78‡	85.14‡	92.01‡	79.56‡	83.72‡	86.34†	85.29	76.44‡	89.41‡
④	$\text{STAG}^{\text{GB}}_{\text{oracle}}$	95.73	93.78	96.66	96.43	98.60	94.43	94.37	93.48	97.41	94.22	97.91
⑤	$\text{STACK}^{\text{GB}}_{\text{oracle}}$	96.43	98.67	96.87	98.38	98.90	92.62	94.21	96.46	96.55	93.33	98.34
⑥	$\text{STAG}^{\text{GB}}_{\text{GB}}$	84.44	85.83‡	83.68‡	84.91	91.64‡	79.31‡	82.87‡	86.28‡	85.18	75.89‡	88.77‡
⑦	$\text{STACK}^{\text{GB}}_{\text{GB}}$	84.23	85.69†	83.44	84.80	91.49	78.90†	82.64	86.08	85.24	75.39	88.62

Table 6: Results (LAS) for the graph-based parser for different experiments on development sets.

		avg.	ar	eu	fr	de	he	hu	ko	pl	sv	en
①	BL^{TB}	84.04	85.69	82.22	84.07	91.15	78.80	83.27	85.97	84.51	75.65	89.06
②	$\text{STAG}^{\text{TB}}_{\text{GB}}$	84.94	86.06‡	84.02‡	84.68‡	91.98‡	79.82	83.83‡	86.56‡	85.73†	77.20‡	89.48‡
③	$\text{STACK}^{\text{TB}}_{\text{GB}}$	84.86	86.19‡	83.86‡	84.55‡	91.98‡	79.76	83.87‡	86.25	85.65	77.16‡	89.30
④	$\text{STAG}^{\text{TB}}_{\text{oracle}}$	96.66	94.09	97.65	96.64	98.80	95.70	96.46	94.90	98.50	95.65	98.16
⑤	$\text{STACK}^{\text{TB}}_{\text{oracle}}$	96.83	98.80	97.08	98.62	98.65	92.96	96.14	97.27	97.21	93.18	98.42
⑥	$\text{STAG}^{\text{TB}}_{\text{TB}}$	84.16	85.77	82.49	84.05	91.44‡	78.69	83.61‡	85.92	84.84	75.78	89.01
⑦	$\text{STACK}^{\text{TB}}_{\text{TB}}$	84.12	85.73	82.24	84.29†	91.35‡	78.67	83.45†	85.86	84.76	75.66	89.22

Table 7: Results (LAS) for the transition-based parser for different experiments on development sets.

Figure 4 displays the graph-based parser's performance relative to dependency length in terms of precision and recall.[8] Precision is defined as the percentage of correct predictions among all predicted arcs of length l and recall is the percentage of correct predictions among gold standard arcs of length l.[9] In all graphs, the stacked and

[8]The corresponding curves for the transition-based parser look very similar.

[9]For precision, the bin sizes shown as grey bars are averages over all three systems, as the number of predicted arcs of a certain length can vary.

supertagged systems show a consistent improvement over the baseline. Moreover, the curves of the stacked and supertagged systems are mostly parallel and close to each other.

Supertagging and stacking thus do not just appear similar at the macro level in terms of LAS. The analysis shows that their contributions are also very similar when broken down by dependency or sentence length and the improvements are not restricted to sentences or arcs of particular lengths. We therefore conclude that both methods are indeed doing the same thing.

4.3 Oracle Experiments

In order to assess the potential utility of supertags we provided the parsers with gold supertags. We expect the gold supertags to give a considerable boost to accuracy as they encode correct syntactic information. Intuitively, we would expect the corresponding stacking experiment (providing gold trees) to reach 100% accuracy since the parser receives the full solution as features. However, this assumption turns out not to hold.

Row ④ in Tables 6 and 7 shows the results for the supertag experiments. Comparing row ④ with row ②, we find big jumps (between 7 and 20% absolute) in performance. For German, English, and Polish performance goes up even to 97/98%. These huge jumps are due to the amount of syntactic information encoded in the supertags, which is much higher than in POS tags for example.

Row ⑤ in Tables 6 and 7 shows the results for the stacking experiments. Surprisingly, stacking with gold dependency trees does not reach 100% accuracy. Moreover, comparing rows ④ and ⑤ we find that on average supertagging and stacking improve performance of a parser to the same extent.

The fact that gold supertags do not yield maximum accuracy is not so surprising since a supertag sequence does not encode the full dependency tree, but merely indicates direction of heads and dependents. However, it is puzzling that stacking with gold trees does not lead to perfect parsing results. In case of the transition-based parser, the reason might be that the parser does not do exhaustive search but uses beam search to explore only a fraction of the search space. That is, the gold solution can get pruned early enough that the parser never considers it. For the graph-based parser this result is more unexpected since this parser does exact search. We currently do not have any explanation for this, however we hypothesize that the lack of regularization during training might assign enough weight on the regular features such that they can override the few stacking features that convey the correct solution.

4.4 Self-Application

Rows ⑥ and ⑦ in Tables 6 and 7 show experiments where we use the same parser at Level 0 and Level 1. We know from Martins et al. (2008) that self-application, i.e. stacking a parser on its own output, leads to at most tiny improvements, especially compared to a setting with different parsers. Our results corroborate these findings. More interestingly, we find a similar effect for supertagging.[10] This effect demonstrates that it is important that Level 0 and Level 1 use different ways of modeling the data in order to benefit from the combination (cf. Surdeanu and Manning (2010)).

5 Supertagging Without Parsers

One potential advantage of supertagging over stacking is the fact that one can predict supertags without a parser. Most previous work predicts supertags using classifiers or sequence models, which is the standard for tagging problems. As tagging is commonly considered an "easier" task than parsing, one could assume that supertags can be predicted very efficiently using standard sequence labeling algorithms. But sequence labelers would not be able to predict the dependency tree in a stacking setup.

The two parsers that we use in the experiments are indeed unlikely to outperform standard sequence labelers in terms of speed. However, greedy arc-standard parsers are very fast. In the next experiment, we therefore compare a greedy arc-standard parser, which is the transition-based parser without beam search, with MarMoT (see Section 3.1). We follow Ouchi et al. (2014) in adding POS tags and morphological information to the feature model of the sequence labeler.

The purpose of this experiment is two-fold: So far, we predicted supertags by predicting a tree first and then deriving the supertags from that tree. Now we test how our previous results compare to supertags predicted by a sequence labeler, which is

[10]Note that most of the improvements in $\text{STAG}^{\text{GB}}_{\text{GB}}$ are actually statistically significant. However, the difference to BL^{GB} is considerably smaller than in the predicted setting in row ② (avg. difference is 0.28% vs. 0.65% points absolute).

		avg.	ar	eu	fr	de	he	hu	ko	pl	sv	en
①	BL^{GB}	84.16	85.64	83.37	84.78	91.46	78.71	82.60	86.08	85.17	75.24	88.59
②	$\text{STAG}^{\text{GB}}_{\text{SL}}$	84.91	86.24‡	84.33‡	84.89	91.89‡	79.82‡	83.54‡	86.85‡	85.89†	76.62‡	89.06‡
③	$\text{STAG}^{\text{GB}}_{\text{GTB}}$	84.65	86.16‡	83.56	85.02†	91.97‡	79.41‡	83.36‡	86.07	85.29	76.56‡	89.12‡
④	$\text{STACK}^{\text{GB}}_{\text{GTB}}$	84.66	86.24‡	83.57	85.11†	91.97‡	79.50‡	83.22‡	86.22	85.45	76.26†	89.09‡
⑤	BL^{TB}	84.04	85.69	82.22	84.07	91.15	78.80	83.27	85.97	84.51	75.65	89.06
⑥	$\text{STAG}^{\text{TB}}_{\text{SL}}$	84.63	85.81	83.58‡	84.07	91.37†	79.86‡	83.91‡	86.98‡	84.93	76.87†	88.91
⑦	$\text{STAG}^{\text{TB}}_{\text{GTB}}$	84.16	85.70	82.44	84.16	91.31	79.38	83.16	85.91	84.59	76.12	88.81
⑧	$\text{STACK}^{\text{TB}}_{\text{GTB}}$	84.17	85.84†	82.46	84.19	91.22	78.93	83.30	85.77	84.92	76.12	88.95

Table 8: Parsing results (LAS) with a sequence labeler and a greedy transition-based parser on development sets.

the common way of predicting supertags. But furthermore, we want to see how supertagging with a sequence labeler compares to supertagging and stacking with a parser that is equally efficient.

Table 8 gives the result of the experiment. We denote the sequence labeler by **SL** and the greedy parser by **GTB**. Rows ② and ⑥ show that, on average, the parsing performance is not harmed by predicting supertags with the sequence labeler instead of one of the parsers (compare to row ② in Tables 6 and 7). It depends on the individual data set whether the sequence labeler is more useful than one of the parsers or not. The supertags predicted by the sequence labeler improve parsing performance to a similar extent.

The experiments with the greedy parser yield different results for the graph-based and the transition-based parser on Level 1: When the graph-based parser acts as Level 1, the greedy parser is slightly behind the sequence labeler. This holds both for supertagging and stacking experiments (compare row ② to rows ③ and ④), which again suggests that supertagging and stacking are interchangeable. However, when Level 1 is the transition-based parser, we find a self-application effect for the greedy parser, both in supertagging and stacking (rows ⑤ vs. ⑦ and ⑧). This is not surprising since the decoding algorithms in the beam-search and greedy transition-based parser are identical. It simply underlines the importance of having different algorithms in the setup.

5.1 Out-of-Domain Application

The previous experiment shows that the greedy parser at Level 0 gives competitive results compared to a sequence labeler. Having fast predictors available for stacking or supertagging suggests an application where speed matters, e.g. Ambati et al. (2014) propose supertags to improve the performance of fast parsers in a web scale scenario.

As web data can be any kind of text, the question is whether the positive effects of supertagging and stacking are actually preserved in such an out-of-domain setting. To test this, we conduct experiments on the English Web Treebank (Bies et al., 2012) converted to Stanford Dependency format. Models are trained on sections 2-21 from the English Penn Treebank.

	avg.	answ.	email	news.	review	blog
BL^{GB}	76.28	74.09	75.06	76.16	76.32	79.78
$\text{STAG}^{\text{GB}}_{\text{SL}}$	76.82	74.52‡	75.75‡	76.88‡	76.99‡	79.98
$\text{STACK}^{\text{GB}}_{\text{TB}}$	76.93	74.88‡	75.72‡	76.49	77.10‡	80.44‡
BL^{TB}	76.51	74.41	75.16	76.09	76.76	80.13
$\text{STAG}^{\text{TB}}_{\text{SL}}$	76.83	74.37	75.85‡	76.61†	77.06	80.28
$\text{STACK}^{\text{TB}}_{\text{GB}}$	76.83	74.64	75.68‡	76.96‡	77.14‡	81.05‡
BL^{GTB}	74.42	72.32	73.25	74.00	74.73	77.79
$\text{STAG}^{\text{GTB}}_{\text{SL}}$	75.01	72.75	73.86‡	74.88‡	75.33‡	78.24‡

Table 9: Results (LAS) on the English Web Treebank.

The results in Table 9 show consistent improvements on the five genres of the data set both for supertagging and stacking. Both are thus good methods to improve parsing accuracies when parsing out-of-domain data. Since parsing speed also depends on the Level 1 parser, a greedy transition-based parser would be preferable in such an application. Using supertagging with a sequence labeler to provide syntactic information to the greedy parser is then a good choice because it avoids a self-application effect.

The last two rows in Table 9 show the performance when the greedy parser is acting as Level 1. Supertagging improves over the baseline significantly on 4 out of 5 data sets. However, the baseline for the greedy parser is on average about 2% points absolute behind the other two parsers. This loss in accuracy buys a significant speed-up though. The greedy parser is about 29 times faster[11] than the graph-based parser on the English

[11]We report parsing time. Exact runtimes depend on im-

		avg.	ar	eu	fr	de	he	hu	ko	pl	sv	en
①	BL^{GB}	84.16	85.64	83.37	84.78	91.46	78.71	82.60	86.08	85.17	75.24	88.59
②	$\max(\text{STAG}_{\text{SL}}^{\text{GB}}, \text{STACK}_{\text{TB}}^{\text{GB}})$	85.00	86.25‡	84.33‡	85.14‡	92.01‡	79.82‡	83.72‡	86.85‡	85.89†	76.62‡	89.41‡
③	$(\text{STAG}_{\text{SL}}+\text{STACK}_{\text{TB}})^{\text{GB}}$	85.20	86.62‡	84.51‡	85.35‡	92.22‡	79.90‡	84.23‡	86.67‡	85.78†	76.98‡	89.74‡
④	BL^{TB}	84.04	85.69	82.22	84.07	91.15	78.80	83.27	85.97	84.51	75.65	89.06
⑤	$\max(\text{STAG}_{\text{SL}}^{\text{TB}}, \text{STACK}_{\text{GB}}^{\text{TB}})$	84.94	86.19‡	83.86‡	84.55‡	91.98‡	79.86‡	83.91‡	86.98‡	85.65	77.16‡	89.30
⑥	$(\text{STAG}_{\text{SL}}+\text{STACK}_{\text{GB}})^{\text{TB}}$	85.13	86.43‡	84.09‡	84.59‡	92.04‡	79.75†	84.08‡	87.32‡	86.01†	77.30‡	89.67‡

Table 10: Results (LAS) on development sets for combining supertags and stacking.

data set and even 80 times faster on the Arabic data set. As the Arabic data set has very long sentences, the higher complexity of the graph-based parser has a notable effect on its performance. Compared to the beam-search transition-based parser, the greedy parser is about 10 times faster on English and 5 times faster on Arabic.

6 Combining Supertagging and Stacking

We now explore whether the combination of supertagging and stacking yields even better parsers.

In rows ③ and ⑥ in Table 10, we show results when supertag and stacking features come from different sources, i.e. they were predicted by different tools[12]. For both parsers, the sequence labeler predicts the supertags and the respective other parser provides the tree for the stacking features. The combinations are better than the baseline. Rows ② and ⑤ give results from the best single source, i.e. either $\text{STAG}_{\text{SL}}^{\text{Y}}$ or $\text{STACK}_{\text{X}}^{\text{Y}}$.

For most of the languages the difference between the combination and the best single component is statistically not significant, except Arabic, German, Hungarian, and English for $(\text{STAG}_{\text{SL}}+\text{STACK}_{\text{TB}})^{\text{GB}}$, and Arabic for $(\text{STAG}_{\text{SL}}+\text{STACK}_{\text{GB}})^{\text{TB}}$. The increment goes up to 0.51 in case of Hungarian. On average, the gains are, however, marginal – the graph-based parser's accuracy increases by 0.2% absolute and the transition-based parser improves by 0.18% absolute. Although these differences denote improvements, they are not nearly as high as the improvements over the baseline for the single components and it depends on the actual data set whether it is worth the effort.

In Section 4, we argued that supertagging and stacking are similar and the diversity of tools is the more important factor. The improvements by the combination can also be interpreted along these lines: They are caused by using different tools rather than the fact that we are combining the two methods. It is like stacking onto two parsers instead of one.

plementation and hardware. We therefore give relative numbers so the reader gets an impression of the magnitude.

[12]We did experiments with combining supertags and stacking from the same Level 0 tool, however since the features were derived from the same tree the differences compared to stacking only were negligible as expected.

7 Conclusion

In this paper, we have shown that supertagging as a method for providing syntactic features for statistical dependency parsing (Ambati et al., 2014; Ouchi et al., 2014) is a form of stacking. Although supertags do not convey as much information as full trees, they improve dependency parsers to an equal amount. The two methods are thus in principle interchangeable.

Combining stacking and supertagging only gives improvements if different tools are used. In this case, the improvements come from the involvement of different tools rather than their combination. Furthermore, using supertags in a parser that predicted them itself does not lead to improvements. This is in line with findings by Surdeanu and Manning (2010) on stacking, of which supertagging is a variant. Therefore, while it is not so important which method is used, it is important to use different algorithms in these setups.

Finally, we have shown that sequence labelers can be replaced by greedy parsers in supertagging without compromising quality or speed. We applied them in a cross-domain parsing scenario and demonstrated that supertagging and stacking improve parsing also in this setting.

However, there are circumstances where one method might be preferable over the other, for example, when one wants to stack on a slow parser (cf. Øvrelid et al. (2009)). Rather than running the slow parser on every sentence in a stacking setup, it can be run once on some training data. A supertagger can then be trained on this data to provide syntactic information at a fraction of the cost (see Ambati et al. (2014) for CCG).

Acknowledgements

The authors would like to thank the anonymous reviewers for their comments. This work was supported by the Deutsche Forschungsgemeinschaft (DFG) via the SFB 732, projects D2 and D8 (PI: Jonas Kuhn), and the Integrated Research Training Group (MGK) of the SFB 732. Agnieszka Faleńska was additionally funded by the Project "International computer science and applied mathematics for business" at the University of Wrocław co-financed with EU funds within the European Social Fund (POKL.04.01.01-00-005/13).

References

Bharat Ram Ambati, Tejaswini Deoskar, and Mark Steedman. 2013. Using CCG categories to improve Hindi dependency parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 604–609, Sofia, Bulgaria, August. Association for Computational Linguistics.

Bharat Ram Ambati, Tejaswini Deoskar, and Mark Steedman. 2014. Improving Dependency Parsers using Combinatory Categorial Grammar. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics, volume 2: Short Papers*, pages 159–163, Gothenburg, Sweden, April. Association for Computational Linguistics.

Srinivas Bangalore and Aravind K. Joshi. 1999. Supertagging: An Approach to Almost Parsing. *Computational Linguistics*, 25(2):237–265, June.

Ann Bies, Justin Mott, Colin Warner, and Seth Kulick. 2012. English Web Treebank LDC2012T13.

Anders Björkelund and Joakim Nivre. 2015. Non-deterministic oracles for unrestricted non-projective transition-based dependency parsing. In *Proceedings of the 14th International Conference on Parsing Technologies*, pages 76–86, Bilbao, Spain, July. Association for Computational Linguistic.

Anders Björkelund, Özlem Çetinoğlu, Agnieszka Faleńska, Richárd Farkas, Thomas Müller, Wolfgang Seeker, and Zsolt Szántó. 2014. The IMS-Wrocław-Szeged-CIS entry at the SPMRL 2014 Shared Task: Reranking and Morphosyntax meet Unlabeled Data. In *Notes of the SPMRL 2014 Shared Task on Parsing Morphologically-Rich Languages*, Dublin, Ireland, August.

Bernd Bohnet. 2010. Top Accuracy and Fast Dependency Parsing is not a Contradiction. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 89–97, Beijing, China, August. Coling 2010 Organizing Committee.

Tim Buckwalter. 2002. Buckwalter Arabic Morphological Analyzer Version 1.0. *Linguistic Data Consortium, University of Pennsylvania, 2002. LDC Catalog No.: LDC2002L49.*

Stephen Clark and James R. Curran. 2004. The Importance of Supertagging for Wide-coverage CCG Parsing. In *Proceedings of the 20th International Conference on Computational Linguistics*, COLING '04, Stroudsburg, PA, USA. Association for Computational Linguistics.

Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. 2006. Online Passive–Aggressive Algorithms. *Journal of Machine Learning Reseach*, 7:551–585, March.

Rickard Domeij, Ola Knutsson, Johan Carlberger, and Viggo Kann. 2000. Granska-an efficient hybrid system for Swedish grammar checking. In *In Proceedings of the 12th Nordic Conference in Computational Linguistics*.

Mikel L. Forcada, Mireia Ginestí-Rosell, Jacob Nordfalk, Jim ORegan, Sergio Ortiz-Rojas, Juan Antonio Pérez-Ortiz, Felipe Sánchez-Martínez, Gema Ramírez-Sánchez, and Francis M Tyers. 2011. Apertium: A free/open-source platform for rule-based machine translation. *Machine Translation*.

Kilian A. Foth, Michael Daum, and Wolfgang Menzel. 2004. Interactive grammar development with WCDG. In *Proceedings of the ACL Interactive Poster and Demonstration Sessions*.

Kilian A. Foth, Tomas By, and Wolfgang Menzel. 2006. Guiding a Constraint Dependency Parser with Supertags. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 289–296, Sydney, Australia, July. Association for Computational Linguistics.

Yoav Goldberg and Michael Elhadad. 2013. Word Segmentation, Unknown-word Resolution, and Morphological Agreement in a Hebrew Parsing System. *Computational Linguistics*, 39(1):121–160, March.

Aravind K. Joshi and Srinivas Bangalore. 1994. Disambiguation of Super Parts of Speech (or Supertags): Almost Parsing. In *Proceedings of the 15th Conference on Computational Linguistics - Volume 1*, COLING '94, pages 154–160, Stroudsburg, PA, USA. Association for Computational Linguistics.

André Filipe Torres Martins, Dipanjan Das, Noah A. Smith, and Eric P. Xing. 2008. Stacking Dependency Parsers. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 157–166, Honolulu, Hawaii, October. Association for Computational Linguistics.

André Martins, Miguel Almeida, and Noah A. Smith. 2013. Turning on the Turbo: Fast Third-Order Non-Projective Turbo Parsers. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 617–622, Sofia, Bulgaria, August. Association for Computational Linguistics.

Ryan McDonald and Joakim Nivre. 2007. Characterizing the Errors of Data-Driven Dependency Parsing Models. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 122–131, Prague, Czech Republic, June. Association for Computational Linguistics.

Thomas Müller, Helmut Schmid, and Hinrich Schütze. 2013. Efficient Higher-Order CRFs for Morphological Tagging. In *In Proceedings of EMNLP*.

Takashi Ninomiya, Takuya Matsuzaki, Yoshimasa Tsuruoka, Yusuke Miyao, and Jun'ichi Tsujii. 2006. Extremely Lexicalized Models for Accurate and Fast HPSG Parsing. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 155–163, Sydney, Australia, July. Association for Computational Linguistics.

Joakim Nivre and Ryan McDonald. 2008. Integrating Graph-Based and Transition-Based Dependency Parsers. In *Proceedings of ACL-08: HLT*, pages 950–958, Columbus, Ohio, June. Association for Computational Linguistics.

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'09)*, pages 73–76, Paris, France, October. Association for Computational Linguistics.

Joakim Nivre. 2009. Non-Projective Dependency Parsing in Expected Linear Time. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 351–359, Suntec, Singapore, August. Association for Computational Linguistics.

Hiroki Ouchi, Kevin Duh, and Yuji Matsumoto. 2014. Improving Dependency Parsers with Supertags. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics, volume 2: Short Papers*, pages 154–158, Gothenburg, Sweden, April. Association for Computational Linguistics.

Lilja Øvrelid, Jonas Kuhn, and Kathrin Spreyer. 2009. Improving data-driven dependency parsing using large-scale LFG grammars. In *Proceedings of the ACL-IJCNLP 2009 Conference Short Papers*, pages 37–40, Suntec, Singapore, August. Association for Computational Linguistics.

Sangwon Park, Donghyun Choi, Eunkyung Kim, and Keysun Choi. 2010. A plug-in component-based Korean morphological analyzer. In *Proceedings of HCLT 2010*.

Yves Schabes, Anne Abeille, and Aravind K. Joshi. 1988. Parsing Strategies with 'Lexicalized' Grammars: Application to Tree Adjoining Grammars. In *Proceedings of the 12th Conference on Computational Linguistics - Volume 2*, COLING '88, pages 578–583, Stroudsburg, PA, USA. Association for Computational Linguistics.

Helmut Schmid, Arne Fitschen, and Ulrich Heid. 2004. SMOR: A German Computational Morphology Covering Derivation, Composition, and Inflection. In *Proceedings of the IVth International Conference on Language Resources and Evaluation (LREC 2004*, pages 1263–1266.

Djamé Seddah, Reut Tsarfaty, Sandra Kübler, Marie Candito, Jinho Choi, Matthieu Constant, Richárd Farkas, Iakes Goenaga, Koldo Gojenola, Yoav Goldberg, Spence Green, Nizar Habash, Marco Kuhlmann, Wolfgang Maier, Joakim Nivre, Adam Przepiorkowski, Ryan Roth, Wolfgang Seeker, Yannick Versley, Veronika Vincze, Marcin Woliński, Alina Wróblewska, and Eric Villemonte de la Clérgerie. 2014. Overview of the spmrl 2014 shared task on parsing morphologically rich languages. In *Notes of the SPMRL 2014 Shared Task on Parsing Morphologically-Rich Languages*, Dublin, Ireland.

Mihai Surdeanu and Christopher D. Manning. 2010. Ensemble Models for Dependency Parsing: Cheap and Good? In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 649–652, Los Angeles, California, June. Association for Computational Linguistics.

Marcin Woliński. 2006. Morfeusz - A practical tool for the morphological analysis of Polish. In *Intelligent information processing and web mining*, pages 511–520. Springer.

Yue Zhang and Stephen Clark. 2008. A Tale of Two Parsers: Investigating and Combining Graph-based and Transition-based Dependency Parsing. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 562–571, Honolulu, Hawaii, October. Association for Computational Linguistics.

Yue Zhang and Joakim Nivre. 2011. Transition-based Dependency Parsing with Rich Non-local Features. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 188–193, Portland, Oregon, USA, June. Association for Computational Linguistics.

Zhenxia Zhou. 2007. Entwicklung einer französischen Finite-State-Morphologie. Diplomarbeit, Institute for Natural Language Processing, University of Stuttgart.

János Zsibrita, Veronika Vincze, and Richárd Farkas. 2013. Magyarlanc 2.0: Szintaktikai elemzés és felgyorsított szófaji egyértelműsítés. In *IX. Magyar Számítógépes Nyelvészeti Konferencia*.

Author Index