Efficient Parsing of Syntactic and Semantic Dependency Structures

Bernd Bohnet
International Computer Science Institute
1947 Center Street
Berkeley 94704, California
bohnet@icsi.Berkeley.edu

Abstract

In this paper, we describe our system for the 2009 CoNLL shared task for joint parsing of syntactic and semantic dependency structures of multiple languages. Our system combines and implements efficient parsing techniques to get a high accuracy as well as very good parsing and training time. For the applications of syntactic and semantic parsing, the parsing time and memory footprint are very important. We think that also the development of systems can profit from this since one can perform more experiments in the given time. For the subtask of syntactic dependency parsing, we could reach the second place with an accuracy in average of 85.68 which is only 0.09 points behind the first ranked system. For this task, our system has the highest accuracy for English with 89.88, German with 87.48 and the out-of-domain data in average with 78.79. The semantic role labeler works not as well as our parser and we reached therefore the fourth place (ranked by the macro F1 score) in the joint task for syntactic and semantic dependency parsing.

1 Introduction

Depedendency parsing and semantic role labeling improved in the last years significantly. One of the reasons are CoNLL shared tasks for syntactic dependency parsing in the years 2006, 2007 (Buchholz and Marsi, 2006; Nivre et al., 2007) and the CoNLL shared task for joint parsing of syntactic and semantic dependencies in the year 2008 and of cause this shared task in 2009, cf. (Surdeanu et al., 2008; Hajič et al., 2009). The CoNLL Shared Task 2009 is to parse syntactic and semantic dependencies of seven languages. Therefore, training and development data in form of annotated corpora for Catalan, Chinese, Czech, English, German, Japanese and Spanish is provided, cf. (Taulé et al., 2008; Palmer and Xue, 2009; Hajič et al., 2006; Surdeanu et al., 2008; Burchardt et al., 2006; Kawahara et al., 2002).

There are two main approaches to dependency parsing: Maximum Spanning Tree (MST) based dependency parsing and Transition based dependency parsing, cf. (Eisner, 1996; Nivre et al., 2004; McDonald and Pereira, 2006). Our system uses the first approach since we saw better chance to improve the parsing speed and additionally, the MST had so far slightly better parsing results. For the task of semantic role labeling, we adopted a pipeline architecture where we used for each step the same learning technique (SVM) since we opted for the possibility to build a synchronous combined parser with one score function.

2 Parsing Algorithm

We adopted the second order MST parsing algorithm as outlined by Eisner (1996). This algorithm has a higher accuracy compared to the first order parsing algorithm since it considers also siblings and grandchildren of a node. Eisner‘s second order approach can compute a projective dependency tree within cubic time (O(n^3)).

Both algorithms are bottom up parsing algorithms based on dynamic programming similar to the CKY chart parsing algorithm. The score for a dependency tree is the score of all edge scores. The following

Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL): Shared Task, pages 67–72,
Boulder, Colorado, June 2009.

equation describes this formally.

$$score(S,t) = \sum_{\forall(i,j)\in E} score(i,j)$$

The score of the sentence S and a tree t over S is defined as the sum of all edge scores where the words of S are $w_0...w_1$. The tree consists of set of nodes N and set of edges $E = \langle N \times N \rangle$. The word indices $(0..n)$ are the elements of the node set N. The expression $(i,j) \in E$ denotes an edge which is going from the node i to the node j.

The edge score ($score(i,j)$) is computed as the scalar product of a feature vector representation of each edge $\overrightarrow{f_S}(i,j)$ with a weight vector $\overrightarrow{w}$ where i,j are the indices of the words in a sentence. The feature vector f_S might take not only into account the words with indices i and j but also additional values such as the words before and after the words w_i and w_j. The following equation shows the score function.

$$score(i,j) = \overrightarrow{f_S}(i,j) * \overrightarrow{w}$$

Many systems encode the features as strings and map the strings to a number. The number becomes the index of the feature in the feature vector and weight vector. In order to compute the weight vector, we reimplemented the support vector machine MIRA which implements online Margin Infused Relaxed Algorithm, cf. (Crammer et al., 2003).

3 Labeled Dependency Parsing

The second order parsing algorithm builds an unlabeled dependency tree. However, all dependency tree banks of the shared task provide trees with edge labels. The following two approaches are common to solve this problem. An additional algorithm labels the edges or the parsing algorithm itself is extended and the labeling algorithm is integrated into the parsing algorithm. McDonald et al. (2006) use an additional algorithm. Their two stage model has a good computational complexity since the labeling algorithm contributes again only a cubic time complexity to the algorithm and keeps therefore the joint algorithm still cubic. The algorithm selects the highest scored label due to the score function $score(w_i, label) + score(w_j, label)$ and inserts the highest scored label into a matrix. The scores are also used in the parsing algorithms and added to the edge scores which improves the overall parsing results as well. In the first order parsing scenario, this procedure is sufficient since no combination of edges are considered by the parsing algorithm. However, in the second order parsing scenario where more than one edge are considered by the parsing algorithm, combinations of two edges might be more accurate.

Johansson and Nugues (2008) combines the edge labeling with the second order parsing algorithm. This adds an additional loop over the edge labels. The complexity is therefore O(n^4). However, they could show that a system can gain accuracy of about 2-4% which is a lot.

4 Non-Projective Dependency Parsing

The dependency parser developed in the last years use two different techniques for non-projective dependency parsing.

Nivre and Nilsson (2005) uses tree rewriting which is the most common technique. With this technique, the training input to the parser is first projectivized by applying a minimal number of lifting operations to the non-projective edges and encoding information about these lifts in edge labels. After these operations, the trees are projective and therefore a projective dependency parser can be applied. During the training, the parser learns also to built trees with the lifted edges and so indirect to built non-projective dependency trees by applying the inverse operations to the lifting on the projective tree.

McDonald and Pereira (2006) developed a technique to rearrange edges in the tree in a postprocessing step after the projective parsing has taken place. Their *Approximate Dependency Parsing Algorithm* searches first the highest scoring projective parse tree and then it rearranges edges in the tree until the rearrangements does not increase the score for the tree anymore. This technique is computationally expensive for trees with a large number of non-projective edges since it considers to re-attach all edges to any other node until no higher scoring trees can be found. Their argument for the algorithm is that most edges in a tree even in language with lot of non-projective sentences, the portion of non-projective edges are still small and therefore by starting with the highest scoring projective tree, typ-

ically the highest scoring non-projective tree is only a small number of transformations away.

Our experiments showed that with the non-projective Approximate Dependency Parsing Algorithm and a threshold for the improvment of score higher than about 0.7, the parsing accuracy improves even for English slightly. With a threshold of 1.1, we got the highest improvements.

5 Learning Framework

As learning technique, we use Margin Infused Relaxed Algorithm (MIRA) as developed by Crammer et al. (2003) and applied to dependency parsing by McDonald et al. (2005). The online Algorithm in Figure 1 processes one training instance on each iteration, and updates the parameters accordingly.

Algorithm 1: MIRA

$\tau = \{S_x, T_x\}_{x=1}^{X}$ // The set of training data consists
// of sentences and the corresponding dependency trees
$\overrightarrow{w}^{(0)} = 0, \overrightarrow{v} = 0$
for n = 1 **to** N
  for x = 1 **to** X
    w^{i+1} = update w^i according to instance (S_x, T_x)
    $v = v + w^{i+1}$
    $i = i + 1$
  end for
end for
$w = v/(N * X)$

The inner loop iterates over all sentences x of the training set while the outer loop repeats the train i times. The algorithm returns an averaged weight vector and uses an auxiliary weight vector v that accumulates the values of w after each iteration. At the end, the algorithm computes the average of all weight vectors by dividing it by the number of training iterations and sentences. This helps to avoid overfitting, cf. (Collins, 2002).

The update function computes the update to the weight vector w^i during the training so that wrong classified edges of the training instances are possibly correctly classified. This is computed by increasing the weight for the correct features and decreasing the weight for wrong features of the vectors for the tree of the training set $\overrightarrow{f_{T_x}} * w^i$ and the vector for the predicted dependency tree $\overrightarrow{f_{T'_x}} * w^i$.

The update function tries to keep the change to the parameter vector w^i as small as possible for correctly classifying the current instance with a difference at least as large as the loss of the incorrect classifications.

6 Selected Parsing Features

Table 1, 4 and 2 give an overview of the selected features for our system. Similar to Johansson and Nugues (2008), we add the edge labels to each features. In the feature selection, we follow a bit more McDonald and Pereira (2006) since we have in addition the lemmas, morphologic features and the distance between the word forms.

For the parsing and training speed, most important is a fast feature extraction beside of a fast parsing algorithm.

Standard Features	
h-f/l	h-f/l, d-pos
h-pos	h-pos, d-f/l
d-f/l	h-f/l, d-f/l
d-pos	h-pos, d-pos
h-f/l,h-pos	h-f/l, d-f/l, h-pos
d-f/l,d-pos	h-f/l, d-f/l, d-pos
	h-pos, d-pos, h-f/l
	h-pos, d-pos, d-f/l
	h-pos, d-pos, h-f/l, d-f/l

Table 1: Selected standard parsing features. h is the abbrevation for head, d for dependent, g for grandchild, and s for sibling. Each feature contains also the edge label which is not listed in order to save some space. Additional features are build by adding the **dir**ection and the **dist**ance plus the direction. The direction is left if the dependent is left of the head otherwise right. The distance is the number of words between the head and the dependent, if $\leq$5, 6 if $>$5 and 11 if $>$10. $\oplus$ means that an additional feature is built with the previous part plus the following part. f/l represent features that are built once with the form and once with the lemma.

Selected morphologic parsing features.

$\forall$ h-morpheme $\in$ head-morphologic-feature-set **do**
  $\forall$ d-morpheme $\in$ dependent-morphologic-feature-set **do**
    build-feautre: h-pos, d-pos, h-morpheme, d-morpheme

7 Implementation Aspects

In this section, we provide implementation details considering improvements of the parsing and training time. The training of our system (parser) has

Linear Features	Grandchild Features	Sibling Features
h-pos, d-pos, h-pos + 1	h-pos, d-pos, g-pos, dir(h,d), dir(d,g)	d-f/l, s-f/l ⊕ dir(d,s) ⊕dist(d,s)
h-pos, d-pos, h-pos - 1	h-f/l, g-f/l, dir(h,d), dir(d,g)	d-pos, s-f/l ⊕ dir(d,s) ⊕dist(d,s)
h-pos, d-pos, d-pos + 1	d-f/l, g-f/l, dir(h,d), dir(d,g)	d-pos, s-f/l ⊕ dir(d,s)+ ⊕ dist(d,s)
h-pos, d-pos, d-pos - 1	h-pos, g-f/l, dir(h,d), dir(d,g)	d-pos, s-pos ⊕dir(d,s)⊕dist(d,s)
h-pos, d-pos, h-pos - 1, d-pos - 1	d-pos, g-f/l, dir(h,d), dir(d,g)	h-pos, d-pos, s-pos, dir(h,d), dir(d,s) ⊕dist(h,s)
h-f/l, g-pos, dir(h,d), dir(d,g)	h-f/l, s-f/l, dir(h,d), dir(d,s)⊕dist(h,s)	h-pos, s-f/l, dir(h,d), dir(d,s)⊕dist(h,s)
d-f/l, g-pos, dir(h,d), dir(d,g)	d-f/l, s-f/l, dir(h,d), dir(d,s) ⊕dist(h,s)	d-pos, s-f/l, dir(h,d), dir(d,s)⊕dist(h,s)
		h-f/l, s-pos, dir(h,d), dir(d,s)⊕dist(h,s)
		d-f/l, s-pos, dir(h,d), dir(d,s)⊕dist(h,s)

Table 2: Selected Features.

three passes. The goal of the first two passes is to collect the set of possible features of the training set. In order to determine the minimal description length, the feature extractor collects in the first pass all attributes that the features can contain. For each attribute (labels, part-of-speech, etc.), the extractor computes a mapping to a number which is continous from 1 to the count of elements without duplicates.

We enumerate in the same way the feature patterns (e.g. h-pos, d-pos) in order to distinguish the patterns. In the second pass, the extractor builds the features for all training examples which occur in the train set. This means for all edges in the training examples.

We create the features with a function that adds iteratively the attributes of a feature to a number represented with 64 bits and shifts it by the minimal number of bits to encode the attribute and then enumerates and maps these numbers to 32 bit numbers to save even more memory.

Beside this, the following list shows an overview of the most important implementation details to improve the speed:

1. We use as feature vector a custom array implementation with only a list of the features that means without double floating point value.

2. We store the feature vectors for $f(label, w_i, w_j)$, $f(label, w_i, w_j, w_g)$, $f(label, w_i, w_j, w_s)$ etc. in a **compressed** file since otherwise it becomes the bottleneck.

3. After the training, we store only the parameters of the support vector machine which are higher than a threshold of 1×10^{-7}

Table 3 compares system regarding their performance and memory usage. For the shared task, we use the system (3) with integrated labeling.

System	(1)	(2)	(3)
Type	2nd order	2nd order	2nd order
Labeling	separate	separate	integrated
System	baseline	this	this
Training	22 hours	3 hours	15 hours
	7GB	1.5 GB	3 GB
Parsing	2000 ms	50 ms	610 ms
		700 MB	1 GB
LAS	0.86	0.86	0.88

Table 3: Performance Comparison. For the baseline system (1), we used the system of McDonald and Pereira (2006) on a MacPro 2.8 Ghz as well for our implementation (2). For system (3), we use a computer with Intel i7 3.2 Ghz which is faster than the MacPro. For all systems, we use 10 training iterations for the SVM Mira.

8 Semantic Role Labeling

The semantic role labeler is implemented as a pipeline architecture. The components of the pipeline are predicate selection (PS), argument identification (AI), argument classification (AC), and word sense disambiguation (WSD).

In order to select the predicates, we look up the lemmas in the Prob Bank, Nom Bank, etc. if available, cf. (Palmer et al., 2005; Meyers et al., 2004). For all other components, we use the support vector machine MIRA to select and classify the semantic role labels as well as to disambiguate the word senese.

The AI component identifies the arguments of each predicate. It iterates over the predicates and over the words of a sentence. In the case that the score function is large or equal to zero the argument is added to the set of arguments of the predicate in question. Table 5 lists for the attribute identification and semantic role labeling.

The argument classification algorithm labels each

Language	Catalan	Chinese	Czech	English	German	Japanese	Spanish	Czech	English	German
Development Set										
LAS	86.69	76.77	80.75	87.97	86.46	92.37	86.53			
Semantic Unlabeled	93.92	85.09	94.05	91.06	91.61	93.90	93.87			
Semantic Labeled	74.98	75.94	78.07	78.79	72.66	72.86	73.01			
Macro (F1)	80.84	76.48	79.42	84.52	79.56	82.63	79.77			
Test Set								Out-of-domain data		
LAS	86.35	76.51	80.11	@89.88	@87.48	92.21	87.19	76.40	@82.64	@77.34
Semantic Labeled	74.53	75.29	79.02	80.39	75.72	72.76	74.31	78.01	68.44	63.36
Macro (F1)	80.44	75.91	79.57	85.14	81.60	82.51	80.75	77.20	75.55	70.35

Table 4: Syntactic and Semantic Scores. @ indicate values that are the highest scores of all systems.

Features with part-of-speech tags	Features with lemmas	Features with rels
arg, path-len	arg, p-lemma $\oplus$ dir $\oplus$ path-len	arg, a-rel $\oplus$ path-len
arg, p-pos	arg, a-lemma, path, dir	arg, a-pos, p-pos, p-rel $\oplus$ path-len
arg, sub-cat, a-pos, p-pos	arg, p-lemma - 1, a-pos, path-len, dir	arg, p-rel, a-pos, lms-lemma
arg, p-pos, a-pos, a-rmc-pos	arg, p-lemma + 1, a-lemma, path, dir	arg, a-pos, p-pos, a-rel
arg, p-pos, a-pos, a-lmc-pos	arg, p-lemma - 1, a-lemma, path, dir	arg, path-rel
arg, p-pos, a-pos, a-lemma-1	arg, p-lemma - 2, a-lemma, path, dir	arg, p-lemma, a-pos, path-rel
arg, sub-cat, a-lemma, dir, path-len	arg, p-lemma, a-lemma, pathPos, dir	
arg, a-pos, a-lemma + 1	arg, p-lemma, p-lemma + 1	
arg, a-pos, a-lemma + 2	arg, p-lemma, p-lemma - 1, a-pos $\oplus$ dir $\oplus$ path-len	
arg, a-pos, a-lemma-lmc	arg, p-lemma, a-lms-lemma, a-pos $\oplus$ dir $\oplus$ path-len	
arg, p-sub-cat, p-pos $\oplus$ dir	arg, p-lemma, path-len $\oplus$ dir	
arg, p-pos, path, p-parent-lemma	arg, p-lemma, path-len $\oplus$ dir	
arg, p-pos, path, p-parent-pos $\oplus$ dir	arg, a-pos, path	
arg, p-pos, a-pos, familyship(p,a)	arg, p-pos, p-lemma, familyship(p,a)	
arg, path-pos	arg, a-pos, p-lemma, familyship(p,a)	
	arg, p-pos, a-lemma, familyship(p,a)	

Table 5: Argument identification and semantic role labeling Features. p is the abbrivation for predicate and a for argument. For the AI component, the attribute arg is either the value *yes* and *no* and for the SRL component, ars is the role label. *path* is the path in terms of up‘s and down‘s. *pathPos* is a path plus the part-of-speech on the path. *dir* is *left*, if the argument is left of the predicate, *equal* if the predicate and argument are equal, otherwise right. *rmc* is the abbrivation for right most child, *lmc* for left most child, and lms left most sibling. *familiship(x,y)* is a function that computes the relation between two words: self, parent, child, ancestor, decendant and none.

identified argument with a semantic role label. The argument classification algorithm selects with a beam search algorithm the combination of arguments with the highest score.

The last component of our pipeline is the word sense disambiguation. We put this against the intuition at the end of our pipeline since experiments showed that other components could not profit from disambiguated word senses but on the other hand the word sense disambiguation could profit from the argument identification and argument classification. In order to disambiguate, we iterate over the words in the corpus that have more than one sense and take the sense with the highest score.

The average time to execute the SRL pipeline on a sentence is less than 0.15 seconds and the training time for all languages less than 2 hours.

9 Conclusion

We provided a fast implementation with good parsing time and memory footprint. Even if we traded off a lot of the speed improvement by using a more expensive decoder and more attributes to get a higher accuracy.

For some languages, features are not provided or the parser does not profit from using these features. For instance, the English parser does not profit from the lemmas and the Chinese as well as the Japanese

corpus does not have lemmas different from the word forms, etc. Therefore, a possible further accuracy and parsing speed improvement would be to select different features sets for different languages or to leave out some features.

Acknowledgments

This work was supported by the German Academic Exchange Service (DAAD). We gratefully acknowledge this support.

References

Sabine Buchholz and Erwin Marsi. 2006. CoNLL-X Shared Task on Multilingual Dependency Parsing. In *In Proc. of CoNLL*, pages 149–164.

Aljoscha Burchardt, Katrin Erk, Anette Frank, Andrea Kowalski, Sebastian Pado, and Manfred Pinkal. 2006. The SALSA corpus: a German corpus resource for lexical semantics. In *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC-2006)*, Genoa, Italy.

Michael Collins. 2002. Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. In *EMNLP*.

Koby Crammer, Ofer Dekel, Shai Shalev-Shwartz, and Yoram Singer. 2003. Online Passive-Aggressive Algorithms. In *Sixteenth Annual Conference on Neural Information Processing Systems (NIPS)*.

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

Jan Hajič, Jarmila Panevová, Eva Hajičová, Petr Sgall, Petr Pajas, Jan Štěpánek, Jiří Havelka, Marie Mikulová, and Zdeňk Žabokrtský. 2006. Prague Dependency Treebank 2.0.

Jan Hajič, Massimiliano Ciaramita, Richard Johansson, Daisuke Kawahara, Maria Antònia Martí, Llu'is Màrquez, Adam Meyers, Joakim Nivre, Sebastian Padó, Jan Štěpánek, Pavel Straňák, Miahi Surdeanu, Nianwen Xue, and Yi Zhang. 2009. The CoNLL-2009 Shared Task: Syntactic and Semantic Dependencies in Multiple Languages. In *Proceedings of the 13th CoNLL-2009, June 4-5*, Boulder, Colorado, USA.

Richard Johansson and Pierre Nugues. 2008. Dependency-based syntactic–semantic analysis with PropBank and NomBank. In *Proceedings of the Shared Task Session of CoNLL-2008*, Manchester, UK.

Daisuke Kawahara, Sadao Kurohashi, and Kôiti Hasida. 2002. Construction of a Japanese relevance-tagged corpus. In *Proceedings of the 3rd International Conference on Language Resources and Evaluation (LREC-2002)*, pages 2008–2013, Las Palmas, Canary Islands.

Ryan McDonald and Fernando Pereira. 2006. Online Learning of Approximate Dependency Parsing Algorithms. In *In Proc. of EACL*, pages 81–88.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005. Online Large-margin Training of Dependency Parsers. In *Proc. ACL*, pages 91–98.

Ryan McDonald, Kevin Lerman, Koby Crammer, and Fernando Pereira. 2006. Multilingual Dependency Parsing with a Two-Stage Discriminative Parser. In *Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 91–98.

Adam Meyers, Ruth Reeves, Catherine Macleod, Rachel Szekely, Veronika Zielinska, Brian Young, and Ralph Grishman. 2004. The nombank project: An interim report. In A. Meyers, editor, *HLT-NAACL 2004 Workshop: Frontiers in Corpus Annotation*, pages 24–31, Boston, Massachusetts, USA, May 2 - May 7. Association for Computational Linguistics.

Joakim. Nivre and Jens Nilsson. 2005. Pseudo-Projective Dependency Parsing. In *In Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics*, pages 99–106.

Joakim Nivre, Johan Hall, and Jens Nilsson. 2004. Memory-Based Dependency Parsing. In *Proceedings of the 8th CoNLL*, pages 49–56, Boston, Massachusetts.

Joakim Nivre, Johan Hall, Sandra Kübler, Rayn McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. 2007. The conll 2007 shared task on dependency parsing. In *Proc. of the CoNLL 2007 Shared Task. Joint Conf. on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, June.

Martha Palmer and Nianwen Xue. 2009. Adding Semantic Roles to the Chinese Treebank. *Natural Language Engineering*, 15(1):143–172.

Martha Palmer, Paul Kingsbury, and Daniel Gildea. 2005. The Proposition Bank: An Annotated Corpus of Semantic Roles. volume 31, pages 71–106.

Mihai Surdeanu, Richard Johansson, Adam Meyers, Lluís Màrquez, and Joakim Nivre. 2008. The CoNLL-2008 shared task on joint parsing of syntactic and semantic dependencies. In *Proceedings of the 12th CoNLL-2008*.

Mariona Taulé, Maria Antònia Martí, and Marta Recasens. 2008. AnCora: Multilevel Annotated Corpora for Catalan and Spanish. In *Proceedings of the LREC-2008*, Marrakesh, Morroco.