Reproducing Czech Syntactic Parsing Results Published in CoNLL Tasks

Lucia Kocincová

NLP Centre, Faculty of Informatics,
Masaryk University, Botanická 68a, 602 00 Brno, Czech Republic
lucyia@mail.muni.cz

Abstract. In this paper, I describe the approach on reproducing MST Parser and MaltParser results for Czech dependency parsing which were published in several tasks of Conference on Computational Natural Language Learning. Firstly, I briefly describe basic principles of parsers. Then, I include features and options that need to be optimized while using the parsers to get the desired results on testing files as well as evaluation methodology used in my research. I also shortly mention hardware requirements for those, who would like to train their own model for Czech language parsing. Finally, I include the practical application of trained models for our approach.

Key words: syntactic analysis, parsing, Czech, parser evaluation

1 Introduction

Dependency parsing universally is nowadays very evolving field in natural language processing, as dependency parsed data are further used for higher layer analysis of natural language. Also, development of tagged treebanks enabled to analyse languages another way – using data-driven parsing. That is why every improvement is proudly presented and various workshops with NLP tasks are founded. One of the best-known is Conference on Computational Natural Language Learning (CoNLL, now part of Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, shortly EMNLP-CoNLL), where shared tasks are hold to challenge the participants in natural language learning systems. Each team is given the same training data, so then the evaluation results can be better compared.

2 CoNNL shared task results for Czech language

Czech language dependency parsing was part of CoNLL shared task in 2006 [1], 2007 [2] and 2009 [3] and the aim of this paper is to describe the process of the approach to get the results shown in Table 1, although it may not be possible to achieve the exact accuracy, as it is covered later in the chapter.

Aleš Horák, Pavel Rychlý (Eds.): Proceedings of Recent Advances in Slavonic Natural Language Processing, RASLAN 2012, pp. 15–22, 2012.

Table 1. Results presented in CoNNL shared tasks.

best accuracy in year	**Labeled Accuracy**	**Unlabeled Accuracy**
2006	80.18	87.30
2007	80.19	86.28
2009	80.38	(not tested)
MaltParser (2006)	78.42	84.80
MST Parser (2006)	80.18	87.30

2.1 Reasons why results may not be reproduced exactly

At the beginning of the research, I was determined that I should simulate all conditions that were depicted in conference papers but after further study of various materials, my determination changed. There were decisive reasons that cause the fact that my results may, or precisely, can not be the same as gained in CoNNL tasks:

- different scoring metrics were used (e.g. punctuation was not counting)
- versions of parsers were not stated
- different training and/or evaluation data (e.g. old version of corpora) may be used[1]
- different data format was used in 2009

In addition, I decided to use the latest versions of parsers because of the fact, that trained models from old versions are no longer runnable under the new ones. Therefore, old version models could be counterproductive if parsing data for later applications, such identified in section 7.

3 MaltParser

MaltParser is a complex system for statistical dependency parsing developed by Johan Hall, Jens Nilsson and Joakim Nivre at Växjö University and Uppsala University in Sweden. Using MaltParser, an induced model can be generated from corpus and then this model can be used to parse new data [4].

Latest version 1.7.2 was released on 25th September 2012 and is distributed under open source licence[2]. The system is being developed in Java from version 1.0.0.

MaltParser's specification can be divided into three parts[3]:

- **Parsing algorithm**: is defined by a *transition system* which derives dependency trees, together with an *oracle* that is used for reconstruction of each valid transition sequence for dependency structures. Together, there are seven deterministic parsing algorithms:

[1] due to licence conditions, they are not available online

[2] full specification can be found at `http://www.maltparser.org/license.html`

[3] complete list of features with their options can be found at `http://www.maltparser.org/userguide.html`

- **Nivre's algorithm**: is a linear-time algorithm for projective dependency structures that can be run in two modes, either arc-eager (nivreeager) or arc-standard (nivrestandard).
- **Stack algorithm**: are a group of three different algorithms: for projective (stackproj) and non-projective trees (stackeager, stacklazy)
- **Covington's algorithm**: is a quadratic-time algorithm for unrestricted dependency structures which modes can be restricted to projective structures (covproj) or non-projective (covnonproj) structures.

The last two parsing algorithms available in parser are:

- **Planar algorithm**: is another linear-time algorithm but for planar dependency structures (ones that do not contain any crossing links)
- **2-Planar algorithm**: also linear-time algorithm but can parse 2-planar dependency structures (ones whose links may be crossed following a specific rule)

After varying parser algorithms, especially first ones mentioned above, Lazy Stack algorithm carried out best accuracies with both types of learning packages.

– **Learning algorithm**: parser includes two machine learning packages – LIBSVM, a type of support vector machines with kernels, and LIBLINEAR, a type of various linear classifiers learner.
I tried various experiments with both packages to search which one gives best results in meaning of time, memory use and accuracy. As can be seen in Table 3, LIBSVM attained best accuracy and learning time was nearly 73 hours.
– **Feature model**: is an external XML file that specifies features of partially built dependency structure together with main data structures in the parser configuration. Default model, which depends on combination of machine learning package and parsing algorithm, can be also used.
I experimented with a few feature models, either build in or obtained by MaltOptimizer[4]. The best performance in my experiment was accomplished by one made with MaltOptimizer.

For achieving state-of-the-art results, optimization of MaltParser is necessary. It is a non-trivial task because not only machine learning algorithm needs to be correctly configured but also various parsers adjustments require setup - I used splitting functionality for speeding up training and parsing time in case of LIBSVM (split column was CPOSTAG, split structure was Stack[0] and split threshold 1000). In addition, each learning algorithm has options itself that can be enabled [7].

Hardware issues when using MaltParser mainly depend on which learning algorithm is chosen – both can give state-of-art accuracy but in case of LIBSVM, less memory is required, whereas LIBLINEAR is much more faster than LIBSVM (for training and also for parsing, ranging from 2–5 hours for LIBLINEAR and 8–72 hours for LIBSVM).

[4] MaltOptimizer is a free available tool for better and easier optimization of MaltParser, online at http://nil.fdi.ucm.es/maltoptimizer/

4 MSTParser

MSTParser is a non-projective dependency parser based on searching maximum spanning trees over directed graphs and is being developed by Jason Baldrige and Ryan McDonald.

Parser from latest versions supports CoNNL format but it distinguishes in some minor extent when comparing the input data format which accepts MaltParser. The system can parse data in its own MST format which is much more simpler than CoNNL:

w_1	w_2	w_3	...	w_n	– n words of a sentence
p_1	p_2	p_3	...	p_n	– POS tags for each word
l_1	l_2	l_3	...	l_n	– labels of the incoming edge to each word
d_1	d_2	d_3	...	d_n	– position of each words parent

Each sentence in the format is represented by first three or all four lines, where each data is tab spaced and whole sentences are space separated.

Optimization of MSTParser includes options such as number of iteration epochs for training, specifying type of structures for setting parsing algorithm (either projective or non-projective), denoting order of features and option if punctuation should be included in Hamming loss calculation. In newer versions, it is possible to add confidence scores per-edge that mark the parser's confidence in correctness of a particular edge.

While experimenting, I also tried out configuration stated in distribution of MSTParser, but I achieved best accuracy using value 3 for training-k[5] and using more memory than stated to avoid errors caused by running out of memory.

Latest version 0.5.0 was released on 23th January 2012 and the whole system, developed in Java, is distributed under *Apache License V2.0*[6].

Hardware issues with MSTParser are bound with sufficient memory while training a model, more specifically, the parser need to get a specification of how much heap space can be taken – I used about 15GB of memory while creating forest from training file with 1 503 739 tokens.

5 Data

5.1 Training and evaluation data

For training and evaluation purposes, The Prague Dependency Treebank 2.0 (PDT 2.0) was used. The newer version was chosen, because of the fact, that last two mentioned CoNNL tasks based the training on it. Moreover, updated version is free of various errors, such as spelling mistakes and faults on morfological and analytical layer [6].

[5] as stated in documentation, the k value is for non-projective structures only approximate

[6] definition available at `http://www.apache.org/licenses/LICENSE-2.0.html`

I used data anotated on analytical layer, which can be described with following numbers[7]:

	overall	train data	dtest data	etest data
sentences	87 913	68 495 (77.9 %)	9 270 (10.5 %)	10 148 (11.5 %)
tokens	1 503 739	1 171 191 (77.9 %)	158 962 (10.6 %)	173 586 (11.5 %)

For training numerous models, strictly only train part of the DPT 2.0 was used, so the rest of it – development test (dtest) and evaluation test (etest), was kept as unseen data. The data were manually disambiguated.

Before running the parsers, the format of the data has to be firstly changed to the CoNNL format in which a sentence consists of ten columns where each is tab separated (overview of the column and data meaning can be seen in Table 2) and individual sentences are separated by a blank line.

Table 2. CoNLL format preciselly described

Column #	Name	Definition
1	ID	Token counter (starting at 1 for each new sentence)
2	FORM	Word form or punctuation symbol
3	LEMMA	Lemma of word form or an underscore if not available
4	CPOSTAG	Coarse-grained part-of-speech tag
5	POSTAG	Fine-grained part-of-speech tag or identical to the coarse-grained part-of-speech tag if not available
6	FEATS	Unordered set of syntactic and/or morphological features separated by a vertical bar
7	HEAD	Head of the current token, which is either a value of ID or zero (0) – there may be multiple tokens with an ID of zero
8	DEPREL	Dependency relation to the HEAD – the dependency relation may be meaningfull or simply 'ROOT'
9	PHEAD	Projective head of current token, which is either a value of ID or zero (0)
10	PDEPREL	Dependency relation to the PHEAD

5.2 Evaluation metrics

For evaluation, a script was written following basic metrics that shows real accuracy:

$$UA = \frac{correct_{head}}{all_{head}}$$

[7] exhausting and precise description of PDT 2.0 can be found at http://ufal.mff.cuni.cz/pdt2.0/doc/pdt-guide/en/html/ch03.html

$$LA = \frac{correct_{head} \;\; AND \;\; correct_{label}}{all}$$

Where Unlabeled accuracy (UA) is the ratio between correctly determined head column ($correct_{head}$) and all heads (all_{head}) and Labeled accuracy (LA) is the result of correctly determined head column AND correctly determined label column ($correct_{label}$) at the same row over all rows in evaluation data.

6 Results

I can bring some relevant results that show the approach of obtaining the results of CoNNL shared tasks is completed. Table 3 presents selected various combinations of learning and parsing algorithms with results for each accuracy, labeled and unlabeled achieved on dtest. Parsing algorithms in italics cover non-projective dependency structures.
In my experiment, I managed to successfully reproduce MaltParser results – I achieved even higher score in both metrics, labeled and unlabeled accuracy. However, results with MSTParser were not accomplished, as shows Table 4.

Table 3. Accuracy achieved with MaltParser on dtest so far

learning alg.	**parsing alg.**	**Unlabeled accuracy**	**Labeled Accuracy**
LIBLINEAR	nivrestandard	70.62	64.73
	covproj	71.43	80.13
	stackproj	79.67	73.99
	covnonproj	80.58	74.95
	stackeager	82.54	77.14
	stacklazy	83.17	77.74
LIBSVM	nivreeager	83.21	78.42
	nivrestandard	81.51	76.37
	stackproj	83.00	77.47
	stacklazy	**85.02**	**80.05**

Table 4. Accuracy achieved with MSTParser on dtest

Unlabeled accuracy	**Labeled Accuracy**
77.73	69.19
83.01	75.34
83.04	**75.39**

7 Further experiments and practical applications

Further experiments will follow this attempt by turning the parsers for even better performance as recently, higher accuracy (about 2% higher in meaning of LA and 1% of meaning of UA) was published with MaltParser [8] and MST-Parser [9].
The aim of the approach was not only to get the results but it is far more practical. Systems with trained models that got the best accuracy will be used for parsing corpora that will be further utilized for application in SketchEngine[8] which is a corpus query system used by various people, including lexicographers, computer linguists and researchers.

Acknowledgments

This work has been partly supported by the Ministry of the Interior of CR within the project VF20102014003 and by the Czech Science Foundation under the project P401/10/0792.

References

1. Buchholz, S., Marsi E.:CoNLL-X Shared Task on Multilingual Dependency Parsing, In: Proceedings of the Tenth Conference on Computational Natural Language Learning, pp. 149–164 (2006), published: Stroudsburg, PA, USA, online at http://dl.acm.org/citation.cfm?id=1596276.1596305
2. Nivre, J., Hall, J., Kübler, S., McDonald, R., Nilsson, J., Riedel, S., Yuret, D.,: The CoNLL 2007 Shared Task on Dependency Parsing, In: Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007, published: Association for Computational Linguistics, Prague, Czech Republic, pp. 915–932 (2007), online at http://www.aclweb.org/anthology/D/D07/D07-1096
3. Hajič, J., Ciaramita, M., Johansson, R., Kawahara, D., Martí, M. A., Marquez, L., Meyers, A., Nivre, J., Padó, S., Štepánek, J., Straňák, P., Surdeanu, M., Xue, N., Zhang, Y.: The CoNLL-2009 shared task: Syntactic and semantic dependencies in multiple languages. In Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task, pp. 1-18 (2009)
4. Nivre, J., Hall, J., Nilsson, J.: MaltParser: A data-driven parser-generator for dependency parsing, In: Proceedings of LREC-2006, pp. 2216–2219 (2006)
5. McDonald, R., Pereira, F., Ribarov, K., Hajič, J.: Non-projective dependency parsing using spanning tree algorithms, In: Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing, pp. 523–530 (2005)
6. Hajič, J.: Complex Corpus Annotation: The Prague Dependency Treebank, published: Jazykovedný ústav Ľ. Štúra, SAV, Bratislava, Slovakia, 2004
7. Nivre, J., Hall, J.: A Quick Guide to MaltParser Optimization, online at http://maltparser.org/guides/opt/quick-opt.pdf

[8] http://sketchengine.co.uk/

8. Nivre, J. : Non-Projective Dependency Parsing in Expected Linear Time. In: Proceedings of the 47th Annual Meeting of the ACL and the 4th IJCNLP of the AFNLP, pp. 351-359. Association for Computational Linguistics, Suntec, Singapore. (2009)
9. Novák, V., Žabokrtský Z.: Feature Engineering in Maximum Spanning Tree Dependency Parser. In: Proceedings of the 10th International Conference on Text, Speech and Dialogue. Západočeská univerzita, Plzeň, Czechia. Springer-Verlag Berlin Heidelberg, LNCS 4629. (2007)