Preliminary Experiments in Polish Dependency Parsing*

Alina Wróblewska and Marcin Woliński

Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland
{alina.wroblewska,wolinski}@ipipan.waw.pl

Abstract. Preliminary experiments presented in this paper consist in the induction and evaluation of a dependency parser for Polish. We train data-driven dependency models with publicly available parser-generation systems (MaltParser and MSTParser) given a converted dependency structure bank for Polish. Induced Polish dependency parsers are evaluated against a set of gold standard dependency structures using labelled and unlabelled accuracy metrics.

Keywords: dependency parsing, Polish parsing, MaltParser, MSTParser

1 Introduction

In recent years two shared tasks on multilingual dependency parsing have been organised at the Conference on Computational Natural Language Learning (CoNLL 2006 [3] and CoNLL 2007 [13]). Different languages were represented in these tasks, among other some Slavic languages such as Slovene, Bulgarian and Czech. Polish has not been represented in any of these tasks. What is more, dependency parsing is hardly represented by the Polish NLP community[1] and we are not aware of any experiments with data-driven Polish dependency parsing. According to our knowledge, no publicly available Polish dependency parser exists, even if it would be an useful tool in different language processing domains. The predicate-argument structure transparently encoded in dependency-based syntactic representations may be useful in machine translation, question answering or information extraction. As mentioned NLP applications are in the early development stage in Polish, a well performing dependency parser may contribute to improve their quality.

The main goal of our preliminary experiments presented in this paper is the induction and evaluation of a dependency parser for Polish. In order to induce a Polish dependency parser we will proceed according to the following procedure.

* This research is supported by the POIG.01.01.02-14-013/09 project which is co-financed by the European Union under the European Regional Development Fund.

[1] The only dependency parser we are aware of was developed by Obrębski [16, 17]. However, this rule-based parser was only tested against a small artificial test set and no wide-coverage grammar seems to accompany the work. An interesting element of Obrębski's thesis is a proposition of a set of relation labels for Polish.

First, we start with the conversion of an existing Polish constituency treebank [21] into dependency-based representations that constitute our training corpus. Second, labelled dependency graphs serve for training and optimising a parsing model which our dependency parser will be based on. The data-driven dependency model training is performed with two freely available parser-generation systems: MaltParser [14] and MSTParser [9]. Finally, we evaluate trained Polish dependency parsers and compare them in terms of parsing accuracy.

Our paper is structured as follows. Section 2 introduces the constituency treebank of Polish and describes its conversion into a Polish dependency bank. In Section 3 we present parsing systems used in our experiments. Section 4 describes the experimental methodology. Section 5 reports achieved results and compares induced dependency parsers. Section 6 concludes with some ideas for future research.

2 Training Data for our Experiments

2.1 A Constituency Treebank of Polish

A bank of constituency trees for Polish is under preparation at the Institute of Computer Science PAS [21]. The planned size of the bank is 20,000 sentences taken from the balanced hand-annotated subcorpus of the National Corpus of Polish (NKJP, [18]). However, as the project is still ongoing, we have only available trees for about 5000 sentences.

The treebank is being developed in a semi-automatic manner. Candidate parse trees are generated by the parser Świgra [22] and one tree is selected and validated by human annotators. The project uses a new version of Świdziński's formal definition of Polish [20]. The treebank and the grammar are developed in parallel: the feedback given by annotators is used to improve the grammar, which leads to regenerating some trees in the treebank.

The project constructs constituency trees, however, their structure is designed with convertibility in mind. In particular each constituent has its syntactic centre marked, which enables us to convert the trees into dependency structures.

Due to the method of construction the current treebank is biased by the incomplete grammar. It consists only of the sentences accepted by the current version of the grammar. As the grammar is being enriched the bias will be reduced. In consequence it currently makes no sense to test our trained parsers on general text, and so we only use cross validation for evaluation. However, we think it is a good idea to start experiments early, so that any deficiencies leading to problems with conversion to dependency structures can still be cured in the source treebank.

2.2 Conversion of Constituency Trees into Dependency Trees

The process aims at converting the source treebank to a bank of labelled dependency structures. The dependency structure of a sentence is a graph with arcs representing directed binary relations between lexical nodes (tokens). One of related tokens is regarded as a *head* of the dependency relation, while the other one is its *dependent*. Arcs linking lexical nodes are named with dependency labels. We have predefined 29 fine-grained dependency labels in the Polish language. The most important dependency types are represented by subject (*subj*), different objects[2], object in dative (*obj_th*), different obliques[3], sentence predicate (*pred*), adjunct (*adj*), relative clause (*crel*). We also distinguish some incidental dependencies: apposition (*app*), predicative argument (*pd*) and pseudo-dependencies representing agglutinative affixes (*aglt*), auxiliaries (*aux*), complementizer of a finite clausal object (*cobj_fin_form*), conjuncts of a coordination (*conjunct*), conjunction (*coord_form*), punctation mark conjunction (*coord_punct*), negation particles (*neg*), interrogative pronoun (*pron_int*), punctuation marks (*punct*), reflexive markers (*refl*). An example of the Polish dependency structure is given in Figure 1.

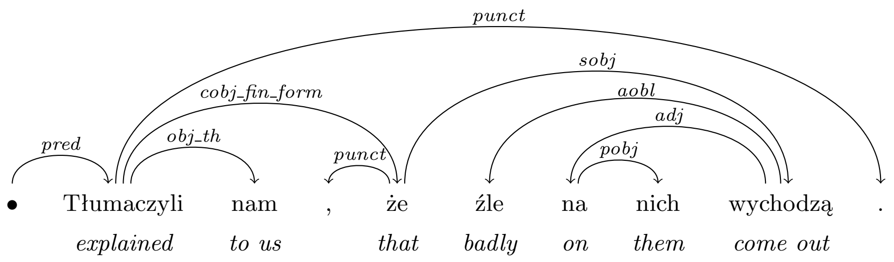


Fig. 1. Dependency structure of the sentence *Tłumaczyli nam, że źle na nich wychodzą.* eng. 'They explained to us that they come out badly on them (photographs).'

Converted dependency structures are stored in the column-based data format of CoNLL shared task [3]. Ten columns contain following data: `ID` (integer token identifier), `FORM` (word form or punctuation symbol), `LEMMA` (lemma of a word form), `CPOSTAG` (coarse-grained part-of-speech tag), `POSTAG` (fine-grained part-of-speech tag), `FEATS` (set of syntactic and/or morphological features separated by a vertical bar), `HEAD` (head `ID` of the current token), `DEPREL` (dependency relation label to the `HEAD`), `PHEAD` (projective head `ID` of the current token) and `PDEPREL` (dependency relation label to the `PHEAD`). All mentioned token attributes except for `PHEAD` and `PDEPREL` are represented in converted dependency structures. If a value is not available from the constituency treebank, an underscore is used as a default value. Since the underlying tagset makes no dis-

[2] Object arguments: indirect question object (*cobj_fin*), infinitival clausal object (*cobj_inf*), object of a numeral (*nobj*), (direct) object (*obj*), adjectival/adverbial argument of a preposition (*paobj*), nominal argument of a preposition (*pobj*), argument of a subordinatig conjunction (*sobj*), verbal object (*vobj*).

[3] Oblique arguments: adverbial oblique argument (*aobl*), prepositional oblique argument of an adjective/adverb (*apobl*), nominal oblique argument(*obl*), prepositional oblique argument (*pobl*).

tinction between `CPOSTAG` and `POSTAG`, we currently use `CPOSTAG` values identical to `POSTAG`.

Converted dependency structures stored in the CoNLL data format are basis of our further experiments.

3 Dependency Parsers

Shared tasks on multilingual dependency parsing at CoNLL 2006 [3] and CoNLL 2007 [13] arouse interest in data-driven dependency parsing. The interest has turned into development of different methods for data-driven dependency parsing. Two of them have dominated other methods: *transition-based* dependency parsing and *graph-based* dependency parsing. A transition-based dependency parser uses a deterministic parsing algorithm that builds a dependency structure of an input sentence based on transitions (shift-reduce actions) predicted by a classifier. The classifier learns to predict the next transition given training data and the parse history. A graph-based dependency parser, in turn, induces parameters of a parsing model over substructures of a dependency graph. The parser learns to score correct trees higher than incorrect ones given an annotated input. The parser finds the highest scored dependency tree. Transition-based and graph-based dependency parsing methods have been implemented as *MaltParser* [14] and *MSTParser* [7], [9], respectively. Even if the considered parsers use different parsing methods, they have achieved similar results for a wide range of languages as described in [8]. We are going to compare both methods in the realistic scenario of dependency parsing of Polish, which is a language not included in any CoNLL shared task on dependency parsing.

3.1 MaltParser – Transition-Based Dependency Parser

The data-driven parser-generator MaltParser[4] [14] trains a transition-based dependency parser for a language given a portion of annotated data in this language. The architecture of an induced deterministic parser consists of three main components: a **parsing algorithm** deriving a labelled dependency structure from an input sentence, a **feature model** helping in prediction of the next parser action, and a treebank-induced **classifier** deterministically predicting the optimal next action given a feature representation of a parser configuration in the current state.

Parsing algorithm A deterministic parsing algorithm provides a basis both for learning and parsing in the MaltParser system. While learning, an *oracle* module maps every tree in the dependency structure bank to a transition sequence that derives this tree, i.e., valid transition sequences are reconstructed from annotated dependency trees. A training oracle is required to train a classifier. While parsing, a transition system builds labelled dependency graphs according

[4] We use MaltParser 1.4.1 downloaded from http://maltparser.org .

to predicted transitions. Transition sequences are predicted by a trained classifier that makes use of a history-based feature model. The *MaltParser* system provides some built-in implementations of parsing algorithms for projective[5] (`nivreeager`, `nivrestandard` and `covproj` [10] or `stackproj` [11]), for non-projective[6] (`covnonproj` [10], `stacklazy` [15], and `stackeager` [11]), and for planar[7] (`planar` [6]) dependency structures.

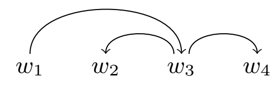
Feature model The history-based feature model is used by MaltParser classifier to predict next actions at non-deterministic choice points given a feature vector. Features are defined in terms of token attributes, i.e., word form (FORM), part of speech (POS), morphological features (FEATS), and lemma (LEMMA) available in input data or dependency types (DEPREL) extracted from partially built dependency graphs and updated during parsing.

Learning algorithm The MaltParser system enables switching between two implementations of machine learning algorithms used to induce a classifier given training data: the LIBSVM library [4] being the implementation of *support vector machines* and the LIBLINEAR package [5] with various *linear classifiers* implemented in it.

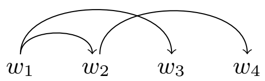
3.2 MSTParser – Graph-based Dependency Parser

A data-driven dependency parser may be trained with the graph-based MSTParser system[8] [7] given a sufficient amount of annotated data. The MST parsing consists in searching for the maximum spanning tree (MST) in a directed graph. MSTParser selects the highest scored dependency tree y as the correct analysis of an input sentence x. The score of the dependency tree is the sum of scores of

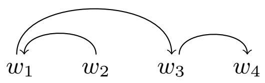
[5] Projective dependency structure – the dependency analysis of a sentence is constrained on the linear word order, i.e., dependency edges are non-crossing with respect to the word order. For example:



[6] Non-projective dependency structure models non-local syntactic constructions such as topicalization, WH-movement, discontinuos NPs, or other resulting in crossing edges, e.g.:



[7] The concept of planarity [19] is similar to the projectivity idea regarding the requirement that dependency links drawn above words in a sentence do not cross. The following structure is planar but not projective because of the root node:



[8] We use MSTParser 0.4.3b downloaded from http://mstparser.sourceforge.net .

all edges in this tree. The score of an edge $s(i,j)$ is defined by [9] as the dot product between a high dimensional feature representation of this edge $f(i,j)$ and a weight vector w learnt during training. The score of a complete dependency tree y for a sentence x is defined as:

$$s(x,y) = \sum_{(i,j)\in y} s(i,j) = \sum_{(i,j)\in y} w \cdot f(i,j)$$

The parser endeavours to find the highest scored dependency tree using one of two parsing algorithms that are available in the MSTParser system: the algorithm of Eisner [7] which deals with the projective structures and the Chu-Liu-Edmonds algorithm [9] which manages non-projective structures. The Eisner parsing algorithm is a bottom-up dynamic programming algorithm with a runtime $O(n^3)$ and the Chu-Liu-Edmonds maximum spanning tree algorithm is a greedy recursive one with the $O(n^2)$ complexity.

Both presented systems are applied to train Polish dependency parsers as described in following sections.

4 Experiments

In this section we present some experiments carried out with MaltParser [14] and MSTParser [9]. Polish is a highly inflected language with flexible word order, what may cause some difficulties for syntactic parser developing. However, both mentioned parsing systems provide an opportunity for adapting parser to characteristic phenomena of a language. We are going to take this opportunity and tune parser parameters, in order to train optimal parsing models.

4.1 MaltParser

Baseline MaltParser The baseline parsing model for Polish is induced using default settings of the MaltParser system. At first, we compare outputs by two baseline parsing models with default settings and different machine learning algorithms which a classifier is trained with. Cross-validation shows that a parser with the LIBLINEAR-classifier performs slightly better (79.5% LAS[9] and 85.8% UAS[10]) than the other one with the LIBSVM-classifier (79.1% LAS and 85.6% UAS). The difference between accuracy of baseline parsing models is not considerable. Neverthless, we have decided to use LIBLINEAR library in our further experiments, as it is faster than the other one. The baseline MaltParser for Polish (default settings and LIBLINEAR classifier) with the labelled attachement score of 79.5% constitutes a point of reference to compare optimized parsing models.

[9] Labelled attachment score (LAS) – the percentage of tokens that are assigned a correct head and a correct dependency type.

[10] Unlabelled attachment score (UAS) – the percentage of tokens that are assigned a correct head.

Experiment 1: Transition System Selection Taking into account that our dependency bank is converted from constituent trees, we only have projective dependency structures. Even if we deal with projective dependency structures, we test out all built-in transition systems, in order to find out the best one applying to the Polish parsing scenario. According to our results (see Table 2 in Section 5), the best performing projection system is `nivrestandard`[11] and the best non-projective system is `stacklazy`. These two transition systems achieve almost the same results with 82.4% LAS and 88.9/89% UAS. Since our data is projective, we have decided to choose the `nivrestandard` projective system for our further experiments.

Experiment 2: Feature Model Estimation The history-based feature model is a combination of static (FORM, POS, LEMMA, FEATS) and dynamic (DEPREL) features. In this experiment we are going to find out the feature combination that improves the parsing performance. We consider the built-in *Nivre-Standard* as the baseline feature model. It is based on FORM, POS and DEPREL features. We expand this baseline model by addition of LEMMA and/or FEATS features. The default and additional features used in the experiment are presented in Table 1.

Table 1. Repertoire of history-based features. Rows correspond to tokens in a parser configuration: `TOP` (the token on the top of a stack), `NEXT` (the next token in the remaining input), `HEAD(TOP)` (the head of `TOP` in a partially built tree), `LDEP(TOP/NEXT)` (the leftmost dependent of `TOP` or `NEXT`), `RDEP(TOP/NEXT)` (the rightmost dependent of `TOP` or `NEXT`). Columns correspond to feature types: FORM (word form), POS (part of speech), DEPREL (dependency relation), LEMMA, FEATS (set of morphological features). ⊕ (default features for the NivreStandard model); + (additional features in the optimised model).

		FORM	**POS**	**DEPREL**	**LEMMA**	**FEATS**
Stack:	`TOP`	⊕	⊕		+	+
Stack:	`TOP-1`		⊕			
Input:	`NEXT`	⊕	⊕		+	+
Input:	`NEXT+1`	⊕	⊕		+	+
Input:	`NEXT+2`		⊕			
Input:	`NEXT+3`		⊕			
Tree:	`HEAD(TOP)`	⊕			+	
Tree:	`LDEP(TOP)`			⊕		
Tree:	`RDEP(TOP)`			⊕		
Tree:	`LDEP(NEXT)`			⊕		
Tree:	`RDEP(NEXT)`			⊕		

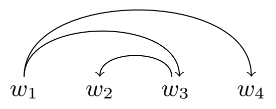
[11] The `stackproj` projective transition system scores just as good as `nivrestandard`. We have decided to use `nivrestandard` in further experiments.

4.2 MSTParser

Baseline MSTParser We induce a baseline parsing model for Polish using default settings of the MSTParser system. The Eisner projective parsing algorithm is running 10 times during training a parsing model. The baseline MSTParser specifies 1-best parses to create constraints and uses first order features.

MSTParser Optimisation As we previously mentioned (see Section 3.2), MSTParser enables switching between two built-in parsing algorithms: the Eisner algorithm and the Chu-Liu-Edmonds algorithm. Our baseline model applies the Eisner parsing algorithm designed for projective dependency structures. Although our training data is projective, we test out the Chu-Liu-Edmonds algorithm designed for non-projective structures as well. As expected, MSTParser with the Chu-Liu-Edmonds algorithm performs slightly worse than the baseline MSTParser (see Table 4).

Hereinafter, we concentrate on the *order-of-features* factor used to score dependency trees. We may choose between *first-order* and *second-order* dependency parsing. In case of the first-order factorization, the score for a dependency graph factors only over single adjacent edges, while in the second-order scoring, the tree score is the sum of adjacent edge pair scores (see Figure 2). The second-order function $s(i, j, k)$ scores a pair of adjacent edges on the same side of the parent node. If the middle argument is ignored, the first-order scoring substitutes the second-order scoring. All adjacent edge pair scores are added up to score the dependency tree.



First-order factorization
$s(1,3) + s(1,4) + s(3,2)$

Second-order factorization
$s(1,-,3) + s(1,3,4) + s(3,-,2)$

Fig. 2. First-order and second-order factorization.

The baseline model applies first-order dependency parsing, as it is a default setting in MSTParser. We are going to test out if the change of the order-of-feature factor influences the Polish MSTParser performance.

The next section reports results of described experiments.

5 Data, Evaluation and Results

The performance of Polish dependency parsers is evaluated with the following metrics: *labelled attachment score* (LAS) and *unlabelled attachment score* (UAS). We evaluate trained parsers in two ways. In the optimisation phase we apply ten-fold cross-validation using the same split of training data for both induced parsers. The final evaluation is performed against the unseen validation data set. We start this section with the presentation of our data and continue with results gained by the Polish MaltParser and the Polish MSTParser.

5.1 Data

The conversion of constituent trees results in 4,601 dependency structures (44,883 tokens). We split the entire dependency bank into a training set with 4,141 sentences (40,346 tokens) and a validation set with 460 sentences (4,537 tokens). As our training data is relatively sparse, we have decided to apply ten-fold cross-validation in the optimisation phase. The validation set is used for the final parser evaluation. It is worth mentioning that sentences in our data set are not long and contain 9.75 tokens on average. Therefore, we suppose to deal with relatively simple syntactic structures in most cases.

5.2 Evaluation of the Polish MaltParser

Experiment 1. Transition System Selection In section 4.1 we gave an account of our first experiment that consists in verification of the built-in transition systems. Results presented in Table 2 show that two parsing models with default parameters and either `nivrestandard` projective system or `stacklazy` non-projective system perform the best (the highest scores are marked in bold). Both parsers perform better than the baseline (`nivreeager`) and their evaluation scores are about 3 percentage points higher. It is also worth noting that we

Table 2. Evaluation of transition systems built in MaltParser. Evaluation metrics: labelled attachment score (LAS) and unlabelled attachment score (UAS).

Parsing Algorithm	**Structures**	**Cross-Validation**	
		LAS	**UAS**
`nivreeager` (baseline)	Projective	79.5	85.8
`nivrestandard`	Projective	**82.4**	**88.9**
`covproj`	Projective	80.3	87.0
`covnonproj`	Non-Projective	80.2	86.8
`stackproj`	Projective	82.3	88.9
`stackeager`	Non-Projective	82.3	88.9
`stacklazy`	Non-Projective	**82.4**	**89.0**
`planar`	Planar	78.6	84.9

get an improvement of parsing performance compared to the baseline parser in case of all built-in transition systems except for `planar` system.

Comparing labelled and unlabelled attachment scores of considered parsing models, we notice that UAS is about 6.5 percentage points higher than LAS. The difference between LAS and UAS may be influenced by relatively small amount of training data.

Experiment 2. Feature Model Estimation The second experiment aims at identification of the optimal feature model. The default feature model consists of word form (FORM), part-of-speech (POS) and dependency relation (DEPREL) features as presented in Table 1. A parser with the default feature model, the `nivrestandard` transition system and the LIBLINEAR classifier achieves 82.4% LAS and 88.9% UAS. Addition of the LEMMA feature to the considered default feature model slightly improves the parser performance, but the difference is below 1 percentage point. Much better improvement is achieved, if the FEATS values are taken into account while predicting the next parser action (4 percentage points above accuracy of the parser with the default feature model). According to results presented in Table 3, the optimal feature model consists of all examined attributes. The parser making use of the optimal feature model achieves 86.9% LAS and 90.2% UAS.

Table 3. Feature model estimation. Evaluation metrics: labelled attachment score (LAS) and unlabelled attachment score (UAS).

Feature Model	**Cross-Validation**		**Final Test**	
	LAS	**UAS**	**LAS**	**UAS**
nivrestandard (FORM, POS, DEPREL)	82.4	88.9	84.2	90.5
nivrestandard + LEMMA	83.1	89.1	84.4	90.6
nivrestandard + FEATS	86.4	90.1	88.0	91.7
nivrestandard + LEMMA + FEATS	**86.9**	**90.2**	**88.8**	**92.2**

We evaluate the best scoring parser with the `nivrestandard` transition system, the LIBLINEAR classifier and the feature model with all attributes against a final validation set (460 dependency structures) and achieve even better results then in cross-validation. The final Polish MaltParser may achieve 88.8% LAS and 92.2% UAS.

5.3 Evaluation of the Polish MSTParser

In the experiment, we try to optimise the induced MSTParser changing two parameters: the parsing algorithm and the order-of-features factor. According to our conjectures, the non-projective Chu-Liu-Edmonds algorithm should not be used to train a dependency parser given projective data. However, the difference between the baseline model and the Chu-Liu-Edmonds parsing model is not significant (about 1 percentage point). Polish admits non-projective structures and they are quite frequent. We suppose that the Chu-Liu-Edmonds parsing algorithm could be a good starting point while training a dependency parser on Polish non-projective data. What is more, results presented in Table 4 show that the order-of-features factor may slightly improve the parsing quality. However, it is true only in cross-validation of MSTParser and not in the final evaluation performed with the unseen test set of 460 sentences. The final Polish MSTParser achieves the 85% LAS and 92% UAS.

Table 4. Evaluation of MSTParser for Polish. Evaluation metrics: labelled attachment score (LAS) and unlabelled attachment score (UAS). Explanation: the Eisner algorithm (Eisner), the Chu-Liu-Edmonds algorithm (Chu-Liu-Edmonds), first-order dependency parsing (1-order), second-order dependency parsing (2-order).

Parameter Settings	**Cross-Validation**		**Final Test**	
	LAS	**UAS**	**LAS**	**UAS**
Eisner + 1-order (baseline)	83.9	90.6	**85.2**	**91.9**
Chu-Liu-Edmonds + 1-order	83.0	89.5	83.8	90.5
Eisner + 2-order	**84.3**	**91.0**	84.4	91.5

5.4 Comparative Error Analysis

The final Polish MaltParser achieves the labelled accuracy score of 88.8% and the unlabelled accuracy score of 92.2%. The best performing Polish MSTParser is our baseline parser at the same time, with 85.2% LAS and 91.9% UAS. It follows that MaltParser has the advantage over MSTParser with regard to the labelled accuracy score (3.6 percentage points). In terms of the unlabelled accuracy score the Polish MaltParser slightly outperforms MSTParser.

We predefined 29 fine-grained dependency labels, which the Polish parsed sentences are annotated with. We evaluate individual labels in terms of *precision*, *recall* and *f-measure*. Automatically labelled dependencies in the final test set are evaluated against the gold standard set. Furthermore, we count the frequency of individual labels in the final set of gold annotated sentences. According to our results, some labels annotated both by MaltParser and by MSTParser have f-score over 90% (*aglt, paobj, cobj_fin_form, cobj_inf, coord_form, neg, pobj, pred, punct, refl, sobj, vobj*). Other labels with the label attachement below 90% (f-score) are listed in Table 5. F-score of about 80% is achieved for following labels: *adj, conjunct* and *nobj*. *pobl* reports f-score above 60% and *cobj_fin* over 40%. Both parsers perform equaly poorly while annotating oblique arguments, especially *aobl* and *apobl*. In case of *aobl*, it is mostly mixed with *adj*, as both of them may be represented by the same part-of-speech tag and morphological features. *apobl*, in turn, is mixed either with *adj* or *pobl* represented by a prepositional phrase. As both *aobl* and *apobl* are sparsely represented in the validation set, they may appear rarely also in the training corpus. The conclusion is that we need more training data to solve this data sparseness problem. The Polish MaltParser considerably outperforms MSTParser in labelling *app, crel, obj_th, obl, pd* and especially in case of *obj* and *subj*, when it achieves over 90% f-score. The Polish MSTParser labels two pseudo-dependency relations more exactly than MaltParser: *coord_punct* and *pron_int*. It is worth noting that we obtain balanced precision and recall values in most cases. If precision and recall values are unequal, than precision value more frequently enhances recall. It follows that if a parser finds a dependency relation between two tokens it is a great chance to label it correctly. The problem is to find all relevant relations what may cause degrade in recall.

Table 5. Labelled accuracy for fine-grained Polish dependency types. Evaluation metrics: precision (Prec), recall (Rec) and f-measure (F).

Dependency	**Frequency**	**MaltParser**			**MSTParser**		
		Prec	**Rec**	**F**	**Prec**	**Rec**	**F**
adj	1304	83.9	82.6	83.2	82.0	82.9	82.4
aobl	17	50.0	17.6	26.1	0.0	0.0	0.0
apobl	9	50.0	11.1	18.2	40.0	22.2	28.6
app	26	65.6	80.8	72.4	75.0	23.1	35.3
cobj_fin	9	44.4	44.4	44.4	75.0	33.3	46.1
conjunct	250	77.5	81.2	79.3	76.0	81.2	78.5
coord_punct	23	62.5	65.2	63.8	67.8	82.6	74.5
crel	18	84.2	88.9	86.5	61.1	61.1	61.1
nobj	31	82.9	93.5	87.9	79.5	100.0	88.6
obj	217	89.5	94.0	91.7	71.9	75.6	73.7
obj_th	41	86.0	90.2	88.1	79.3	56.1	65.7
obl	50	66.7	68.0	67.3	46.1	24.0	31.6
pd	54	89.1	75.9	82.0	71.7	61.1	66.0
pobl	168	67.4	72.6	69.9	68.6	63.7	66.0
pron_int	7	62.5	71.4	66.7	75.0	85.7	80.0
subj	354	93.8	93.8	93.8	75.5	81.1	78.2

6 Conclusions and Future Work

In experiments presented in this paper, dependency parsing models for Polish have been trained using two parser-generators: MaltParser and MSTParser. Both the transition-based MaltParser and the graph-based MSTParser are data-driven systems designed for induction of a dependency parser for a language for which an annotated dependency bank exists. As Polish lacks any dependency bank, we automatically converted trees from the existing constituency treebank into dependency structures.

According to final evaluation results, the Polish MaltParser (89% LAS and 92% UAS) slightly outperformed the Polish MSTParser (85% LAS and 92% UAS) taking their labelled accuracy scores into account. Achieved results are quite good, but we may not forget that our dependency bank contains short sentences (9.7 tokens per sentence on average) which do not have to represent complex syntactic structures.

As no Polish dependency parser is publicly available, the accuracy of dependency parsers for Czech and Russian constitutes our point of reference while evaluating Polish dependency parsers. Czech is one of languages that has participated in CoNLL 2006 and CoNLL 2007. Dependency parsers for Czech were trained on *Prague Dependency Treebank* [2] with 72,700 sentences (1,249,000 tokens and 17.2 tokens per sentence on average) in CoNLL 2006 and with 25,400 sentences (450,000 tokens and 17 tokens per sentence on average) in CoNLL 2007. According to results, the Czech MaltParser achieved 78.4% LAS and 84.8% UAS in the first task and 77.2% LAS and 82.3% UAS in the second CoNLL task. MaltParser was outperformed by the Czech MSTParser with 80.2% LAS and 87.3% UAS.

The Russian MaltParser [12] have been trained on the large dependency treebank SynTagRus [1] containing over 32,000 sentences (460,000 tokens) taken from different genres. The best Russian MaltParser achieved 82.2% LAS and 89.1% UAS. Polish parsers seem to perform better than dependency parsers for Russian and Czech. However, we should take into account simplicity of Polish training data. Even if Polish dependency parsers trained on preselected training data perform well, they may not be appropriate for parsing general text.

We need more training data to cover the bulk of syntactic phenomena of the Polish language. As the Polish constituency treebank is going to be completed in a few months, we will gain four times more training data. The full treebank will be comparable in size to treebanks used for training Czech and Russian dependency parsers. So we expect to be able to achieve a reasonable corpus coverage with parsers trained on the entire corpus. It is an interesting question, however, what accuracy scores can we achieve for that future parser.

In case we need still more training data we plan to explore an alternative solution: an automatic annotation using the projection method. As some well-performing dependency parsers for English exist, they may be applied to analyse the English part of a parallel Polish-English corpus. Using automatic generated word alignment links, English dependencies will be projected onto corresponding Polish tokens. As result, a large Polish dependency bank may be induced. Future studies will show, if a well-performing dependency parser may be trained on a great amount of presumably noisy data.

References

1. Boguslavsky, I., Chardin, I., Grigorieva, S., Grigoriev, N., Iomdin, L., Kreidlin, L., Frid, N.: Development of a Dependency Treebank for Russian and its possible Applications in NLP. In: In Proceedings of the 3rd International Conference on Language Resources and Evaluation, Las Palmas, Gran Canaria. pp. 852–856 (2002)
2. Böhmová, A., Hajič, J., Hajičová, E., Hladká, B.: The PDT: a 3-level annotation scenario. In: Abeillé, A. (ed.) Treebanks: Building and Using Parsed Corpora, Text, Speech and Language Technology, vol. 20, chap. 7. Kluwer Academic Publishers, Dordrecht (2003)
3. 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. CoNLL-X '06, Association for Computational Linguistics (2006)
4. Chang, C.C., Lin, C.J.: LIBSVM: a library for support vector machines (2001), software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm
5. Fan, R.E., Chang, K.W., Hsieh, C.J., Wang, X.R., Lin, C.J.: LIBLINEAR: A library for large linear classification. Journal of Machine Learning Research 9, 1871–1874 (Aug 2008)
6. Gómez-Rodríguez, C., Nivre, J.: A transition-based parser for 2-planar dependency structures. In: Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics. pp. 1492–1501. ACL '10, Association for Computational Linguistics (2010)

7. McDonald, R., Crammer, K., Pereira, F.: Online large-margin training of dependency parsers. In: Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics. pp. 91–98. ACL '05 (2005)
8. McDonald, R., Nivre, J.: 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). pp. 122–131 (2007)
9. McDonald, R., Pereira, F., Ribarov, K., Hajič, J.: Non-projective dependency parsing using spanning tree algorithms. In: Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing. pp. 523–530. HLT '05 (2005)
10. Nivre, J.: Algorithms for deterministic incremental dependency parsing. Computational Linguistics 34, 513–553 (December 2008)
11. Nivre, J.: 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: Volume 1. pp. 351–359. ACL '09, Association for Computational Linguistics (2009)
12. Nivre, J., Boguslavsky, I.M., Iomdin, L.L.: Parsing the SynTagRus treebank of Russian. In: Proceedings of the 22nd International Conference on Computational Linguistics - Volume 1. pp. 641–648. COLING '08 (2008)
13. 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. pp. 915–932. Association for Computational Linguistics, Prague, Czech Republic (June 2007)
14. Nivre, J., Hall, J., Nilsson, J.: Maltparser: a data-driven parser-generator for dependency parsing. In: Proceedings of LREC-2006. pp. 2216–2219 (2006)
15. Nivre, J., Kuhlmann, M., Hall, J.: An improved oracle for dependency parsing with online reordering. In: Proceedings of the 11th International Conference on Parsing Technologies. pp. 73–76. IWPT '09, Association for Computational Linguistics (2009)
16. Obrębski, T.: Automatyczna analiza składniowa języka polskiego z wykorzystaniem gramatyki zależnościowej. Phd thesis, Institute of Computer Science, Polish Academy of Sciences, Warsaw (2002)
17. Obrębski, T.: Mtt-compatible computationally effective surface-syntactic parser. In: Proceedings of First International Conference on Meaning-Text Theory. pp. 259–268. Paris (2003)
18. Przepiórkowski, A., Górski, R.L., Łaziński, M., Pęzik, P.: Recent developments in the National Corpus of Polish. In: Proceedings of the Sixth International Conference on Language Resources and Evaluation, LREC 2010. ELRA, Valetta, Malta (2010)
19. Sleator, D.D., Temperley, D.: Parsing English with a link grammar. In: Third International Workshop on Parsing Technologies. p. 277291 (1993)
20. Świdziński, M.: Gramatyka formalna języka polskiego. Rozprawy Uniwersytetu Warszawskiego, Wydawnictwa Uniwersytetu Warszawskiego, Warszawa (1992)
21. Świdziński, M., Woliński, M.: Towards a bank of constituent parse trees for Polish. In: Sojka, P. (ed.) Text, Speech and Dialogue, 13th International Conference, TSD 2010, Brno, September 2010, Proceedings. LNAI, vol. 6231, pp. 197–204. Springer, Heidelberg (2010)
22. Świgra – an implementation of the formal grammar of Marek Świdziński (2005), http://nlp.ipipan.waw.pl/˜wolinski/swigra/