A Morphosyntactic Brill Tagger for Inflectional Languages

Szymon Acedański[1,2]

[1] Institute of Informatics, University of Warsaw,
ul. Banacha 2, 02-097 Warszawa, Poland,
accek@mimuw.edu.pl,
[2] Institute of Computer Science, Polish Academy of Sciences,
ul. Ordona 21, 01-237 Warszawa, Poland

Abstract. In this paper we present and evaluate a Brill morphosyntactic transformation-based tagger adapted for specifics of highly inflectional languages. Multi-phase tagging with grammatical category matching transformations and lexical transformations brings significant accuracy improvements comparing to previous work. Evaluation shows the accuracy of 92.44% for the Polish language which is higher than the same metric for the other known taggers of Polish: stochastic trigram tagger (90.59%) and hybrid tagger TaKIPI employing decision tree classifier and automatically extracted rule-based tagger used for tagging the IPI PAN Corpus of Polish (91.06%).

Key words: PoS tagger, Brill tagger, inflectional language tagger, morphosyntactic tagger, lexical rules

1 Introduction

Morphosyntactic tagging is a classic problem in NLP with applications in many higher level processing solutions, namely parsing and then information retrieval, speech recognition and machine translation. Part of Speech tagging for English is already well explored and many taggers have been built with accuracy exceeding 98%. In case of inflectional languages these numbers are much lower, reaching 95,78% for Czech [1] and 92.55% for Polish (per [2]; evaluation by Karwańska and Przepiórkowski [3] reports 91.30%).

The most prominent difference between English and inflectional languages is the size of the tagset. Brill [4] uses Brown's Tagset for English, which consists of almost 200 tags, whereas the IPI PAN Polish tagset [5] contains theoretically over 4000 tags and the manually disambiguated part of the IPI PAN Corpus of Polish [6] used for evaluation contains 1054 different tags. The tags for such languages have a specific structure — along with the part of speech, they contain values of grammatical categories appropriate for the particular part of speech (see Table 1 for an example in Polish). Detailed description of the tagset and the meaning of particular grammatical categories can be found in [5].

Table 1. Example tags in Brown's English Tagset and IPI PAN Polish tagset.

English	`VBD`	verb, past tense
	`PPS`	pronoun, personal, nominative, 3rd person singular
Polish	`praet:sg:m1:perf`	l-participle, singular, human masculine, perfective aspect
	`ppron12:sg:nom:f:pri`	1st person pronoun, singular, nominative, feminine

Not only the large tagset makes disambiguation a difficult task, but also free word order in considered languages and even problems of unambiguously defining the correct tags in some cases. Because of this, some corpora allow multiple tags to be assigned to a single segment, whereas other require fully disambiguated tagging, usually providing detailed instructions on how to do this. This matter will be further discussed in the Evaluation section.

Several tagging techniques are commonly known. The most frequently used approaches are: stochastic, e.g., based on Hidden Markov Models [7], and rule-based[3]. Brill [4] presents a transformation-based Part of Speech tagger for English, which automatically chooses good quality transformations given a number of general transformation templates and a training corpus. The tagger used for morphosyntactic disambiguation of the current version of the IPI PAN corpus, called TaKIPI [8], is a hybrid (multiclassifier) transformation-based tagger. Some of the transformations it uses were extracted automatically using machine learning algorithms and then reviewed and adjusted by linguists.

In this paper we describe and evaluate an implementation of the Brill's algorithm, adapted for rich inflectional languages. First steps towards this were described by Acedański and Gołuchowski in 2009 [9], but that tagger was then rewritten with different approaches used in most parts. As in previous work, the adaptation involves splitting the process into phases, so that at first only the part of speech and a few grammatical categories are disambiguated. Remaining categories are determined in the second pass. On top of it, the new, more general approach to transformation templates was developed, and additional transformation templates allowing for transformations which look at particular grammatical categories of surrounding segments were added. Also lexical transformations were used. Finally the tagger was implemented using a new simplified algorithm based on FastTBL [10] and parallelized for better performance.

[3] Throughout this paper, the term *rule-based tagger* is used to denote systems using hand-written rules. For the algorithms involving automatic extraction of rules, the term *transformation-based tagger* is used

2 The original Brill tagger

Let us describe the original Brill's algorithm in some detail. We assume that we are given three corpora — a large high-quality tagged training corpus, smaller tagged corpus called patch corpus and another one — test corpus, which we want to tag. Brill also assumes, that only one correct tag can be assigned to a segment. Let's denote the tag assigned to i-th segment as t_i.

Tagging is performed in four steps:

1. A simple unigram tagger is trained using the large training corpus.
2. The unigram tagger is used to tag the patch corpus.
3. There are certainly some errors in the tagging of the patch corpus. Therefore we want to generate transformations which will correct as many errors as possible.
   (a) We are given a small list of so called *transformation templates*. Brill uses the following templates in his paper:
      i. $t_i := \mathbf{A}$ if $t_i = \mathbf{B} \ \wedge \ \exists_{o \in \mathbf{O_1}} t_{i+o} = \mathbf{C}$,
      ii. $t_i := \mathbf{A}$ if $t_i = \mathbf{B} \ \wedge \ \forall_{o \in \mathbf{O_2}} t_{i+o} = \mathbf{C_o}$,
      iii. $t_i := \mathbf{A}$ if $t_i = \mathbf{B}$ and i-th word is capitalized,
      iv. $t_i := \mathbf{A}$ if $t_i = \mathbf{B}$ and $(i-1)$-th word is capitalized.

      where
      - $\mathbf{O_1} \in \{\{1\}, \{-1\}, \{2\}, \{-2\}, \{1,2\}, \{-1,-2\}, \{1,2,3\}, \{-1,-2,-3\}\}$,
      - $\mathbf{O_2} \in \{\{-2,-1\}, \{-1,1\}, \{1,2\}\}$,
      - $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, $\mathbf{C_o}$ — any tags.

   (b) For each transformation r which can be generated using these templates, we compute two statistics:
      i. $\mathbf{good}(r)$ — the number of places in the patch corpus where the transformation matches and changes an incorrect tag into a correct one,
      ii. $\mathbf{bad}(r)$ — the number of places in the patch corpus where the transformation matches and changes the tagging from correct to incorrect.

   (c) Now we find transformation r_b, which maximizes $\mathbf{good}(r) - \mathbf{bad}(r)$, i.e. reduces the largest possible number of errors when applied. We save the transformation and apply it to the patch corpus. If the patch corpus still contains many errors, return to 3a.
4. The test corpus is first tagged using the unigram tagger, and then the saved transformations are applied in order.

If the test corpus was previously manually tagged, we can evaluate the performance of the tagger.

3 Adaptation for inflectional languages

The algorithm described in the previous section was subsequently extended by applying a number of techniques targeted at improving accuracy of tagging of inflectional languages. These techniques are:

- multi-pass tagging — gradually disambiguating parts of tags,
- generalized transformation templates — allowing for more flexible design and then specific templates for inflectional languages relying on interdependencies between values of grammatical categories,
- lexical transformation templates — allowing to match prefixes and suffixes of processed lexemes,
- simplified implementation of FastTBL algorithm [10] and parallelization of the tagging engine for good performance and maintainability

3.1 Multi-pass tagging

The first technique is used to reduce the size of the transformation space and to avoid too specific transformations in the first stage. It is inspired by [9], where the authors split the tags into two parts sharing only the part-of-speech. In the first run of the Brill tagger the tagset consists of only one of the parts of tags. In the second run, the tagset comprised of the other parts is used, but the previously selected parts of speech are fixed for the second pass. Also Tufiş [11] proposes using a reduced tagset with easily recoverable grammatical categories not present, to improve performance. Out goal is different though — we try to leave some of the hard to disambiguate categories for later stages so that the tagger already has more information from preceding phases.

We consider a sequence T_i ($i \in \{0, \ldots, k-1\}$) of gradually simplified tagsets. T_0 is the original tagset and T_{j+1} ($j \in \{0, \ldots, k-2\}$) are some other tagsets. Projections mapping specific tags to more general tags are also needed: π_j: $T_j \to T_{j+1}$. For each of the tagsets a separate pass of the Brill algorithm is performed. The tag assigned to the i-th segment in the p-th pass ($p \in \{1, \ldots, k\}$) is denoted by t_i^p. In the first pass the simplest tagset T_{k-1} is used. In the p-th pass, for i-th segment, only tags $t_i^p \in T_{k-p}$ are considered such that $\pi_{k-p}(t_i^p) = t_i^{p-1}$.

In our experiments we used only two tagsets — T_0 being the original and T_1 which had information about part of speech, case and person only. π_0 was a natural projection i.e. the one which strips values of grammatical categories not present in T_1. The produced software can be configured for more than two phases with different tagsets.

3.2 Generalized transformation templates

In the original Brill classifier, all the transformation templates are of the following form:

> Change t_i to **A** if it is **B** and ....

In our tagger we generalize the possible transformation templates by allowing other operations than changing the entire tag to be performed. Also, the current tag of a lexeme need not be fully specified in an instantiation of some transformation template.

A particular transformation template consists of a *predicate* template which specifies conditions on the context where the transformation should be applied,

and an *action* template describing the operation to be performed if the predicate matches. For example in the transformation template "change the tag to **A** if the tag is **B**", the first part ("change the tag to **A**") is the action template and the second part ("the tag is **B**") is the predicate template. The same nomenclature is applied to instantiated transformations.

This generalization was performed in order to allow using more general transformations than allowed by the original algorithm. Let's denote by $t_i|_{\text{CASE}}$ the value of grammatical category CASE in the tag of the i-th segment of the text. Now consider the very robust linguistic rule "if an adjective is followed by a noun, then they should agree in CASE".
This rule may be composed of

- an *action*: $t_i|_{\text{CASE}} := t_{i-1}|_{\text{CASE}}$
- a *predicate*: $t_i|_{\text{POS}} = \text{SUBST} \ \wedge \ t_{i-1}|_{\text{POS}} = \text{ADJ}$

The proposed tagger is able to generate transformations resembling such rules. It uses the following predicate templates:

1. $t_i^p = \mathbf{T} \ \wedge \ \exists_{o \in \mathbf{O}} t_{i+o}^p = \mathbf{U}$,
2. $t_i^p = \mathbf{T} \ \wedge \ \forall_{o \in \mathbf{O}} t_{i+o}^p = \mathbf{U_o}$,
3. $t_i^p = \mathbf{T} \ \wedge \ \exists_{o \in \mathbf{O_1}} t_{i+o}^{p-1} = \mathbf{U'}$,
4. $t_i^p|_{\text{POS}} = \mathbf{P} \ \wedge \ t_i^p|_{\mathbf{C}} = \mathbf{X} \quad \wedge \quad \exists_{o \in \mathbf{O}} \left(t_{i+o}^p|_{\text{POS}} = \mathbf{Q} \ \wedge \ t_{i+o}^p|_{\mathbf{C}} = \mathbf{Y} \right)$,
5. $t_i^p|_{\text{POS}} = \mathbf{P} \ \wedge \ t_i^p|_{\mathbf{C}} = \mathbf{X} \quad \wedge \quad \forall_{o \in \mathbf{O}} \left(t_{i+o}^p|_{\text{POS}} = \mathbf{Q_o} \ \wedge \ t_{i+o}^p|_{\mathbf{C}} = \mathbf{Y_o} \right)$,

and action templates:

1. $t_i^p := \mathbf{V}$,
2. $t_i^p|_{\text{POS}} := \mathbf{R}$,
3. $t_i^p|_{\mathbf{C}} := \mathbf{Z}$,

where

- $\mathbf{T}$, $\mathbf{U}$, $\mathbf{U_o}$, $\mathbf{V}$ — any tags valid in pass p,
- $\mathbf{U'}$ — any tag valid in pass $p-1$,
- $\mathbf{P}$, $\mathbf{Q}$, $\mathbf{Q_o}$, $\mathbf{R}$ — any parts of speech valid in pass p,
- $\mathbf{C}$ — any grammatical category valid in pass p,
- $\mathbf{X}$, $\mathbf{Y}$, $\mathbf{Y_o}$, $\mathbf{Z}$ — any values valid for category $\mathbf{C}$,
- $\mathbf{O} \in \{\{1\}, \{-1\}, \{2\}, \{-2\}, \{1,2\}, \{-1,-2\}, \{1,2,3\}, \{-1,-2,-3\}\}$,
- template variables $\mathbf{P}$, $\mathbf{Q}$, $\mathbf{Q_o}$ (for all o at the same time), $\mathbf{R}$, $\mathbf{X}$, $\mathbf{Y}$, $\mathbf{Y_o}$ (for all o at the same time) and $\mathbf{Z}$ could have a special value $\star$ meaning *any*.

Additionally, the actions were implemented in such a way, that they were not applied if they were to assign a tag not reported by the morphological analyzer for a particular segment. In case of actions 2 and 3, the nearest possible tag was used instead. The metric used here is the number of matching values of grammatical categories, but only tags with the expected part of speech are considered. If no such tags are possible, the action is not performed.

3.3 Lexical transformations

Another extension which proved very useful are lexical transformation templates proposed by Brill in a later paper [12]. Megyesi [13] subsequently explored them for Hungarian (which is an agglutinative language, with a number of affixes possessing grammatical functions). The results were very promising. The author used the following predicate templates:

1. $t_i^p = \mathbf{T} \ \wedge$ $orth_i$ contains letter $\mathbf{L}$,
2. $t_i^p = \mathbf{T} \ \wedge$ $orth_i$ starts/ends with $\mathbf{S}$, $|\mathbf{S}| < 7$,
3. $t_i^p = \mathbf{T} \ \wedge$ $orth_i$ with deleted prefix/suffix $\mathbf{S}$, $|\mathbf{S}| < 7$, is a word,
4. $t_i^p = \mathbf{T} \ \wedge \ (orth_{i_1} = \mathbf{W} \vee orth_{i+1} = \mathbf{W})$.

Here $orth_i$ simply denotes the orthographic representation of the i-th segment. Inspired by this work, and after some experiments, we extended the list of predicate templates by only the prefix/suffix matching:

1a. $t_i^p = \mathbf{T} \ \wedge$ $orth_i$ ends with $\mathbf{S}'$

1b. $t_i^p = \mathbf{T} \ \wedge$ $orth_i$ starts with $\mathbf{S}'$

4a. $t_i^p|_{\mathrm{POS}} = \mathbf{P} \ \wedge \ t_i^p|_{\mathbf{C}} = \mathbf{X} \ \wedge$ $orth_i$ ends with $\mathbf{S}$
$\wedge \ \exists_{o \in \mathbf{O}} \big(t_{i+o}^p|_{\mathrm{POS}} = \mathbf{Q} \ \wedge \ t_{i+o}^p|_{\mathbf{C}} = \mathbf{Y} \big)$

4b. $t_i^p|_{\mathrm{POS}} = \mathbf{P} \ \wedge \ t_i^p|_{\mathbf{C}} = \mathbf{X}$
$\wedge \ \exists_{o \in \mathbf{O}} \big(t_{i+o}^p|_{\mathrm{POS}} = \mathbf{Q} \ \wedge t_{i+o}^p|_{\mathbf{C}} = \mathbf{Y}$
$\wedge$ $orth_{i+o}$ ends with $\mathbf{S} \big)$

where $\mathbf{S}$ and $\mathbf{S}'$ are any strings no longer than 3 and 2 characters respectively. This resulted in over 1.5% accuracy improvement over the Brill tagger with only generalized transformations, as tested for Polish.

3.4 Simplified FastTBL implementation

The idea behind the FastTBL algorithm [10] is the minimization of the number of accesses to data structures for storing $good(\cdot)$ and $bad(\cdot)$ functions. Unfortunately this comes with the increased complexity — there are 8 possible branches of execution in the main loop. Therefore we designed a simplified version of this algorithm presented as Algorithm 1. It allows redundant updates of $good(\cdot)$ and $bad(\cdot)$, but this extra work does not significantly influence the total running time of the algorithm, because the most computationally intensive work is generating the possible transformations in lines 2, 20 and 27, as well as the application of the generated transformation in 18.

3.5 Parallelization

Finally, the tagger was implemented specifically for multiprocessing environment, mostly because of the high memory requirements for storing the $good(\cdot)$ and $bad(\cdot)$ functions. The ordinary 32-bit Linux operating system originally used for experiments does not allow for more than 2GB of memory per process. The

Algorithm 1 Pseudocode of the simplified FastTBL algorithm.

```
{Initializing good and bad data structures}
for i = 0 to len(text) do
    for each transformation r which matches at position i do
        if r corrects the classification of i-th segment then
            increase good(r)
        else if r changes the classification of i-th segment from correct to wrong then
            increase bad(r)
        end if
    end for
end for

{Main loop — generating transformations}
loop
    b := arg max_r(good(r) − bad(r)) {the best transformation}
    if good(b) − bad(b) < threshold then
        return
    end if
    add b to the sequence of generated transformations
    text' := text after application of b
    for each position i in vicinity of the changes performed by b do
        for each transformation r which matches at position i in text do
            if r corrected the classification of i-th segment in text then
                decrease good(r)
            else if r miscorrected the classification of i-th segment in text then
                decrease bad(r)
            end if
        end for
        for each transformation r which matches at position i in text' do
            if r corrects the classification of i-th segment in text' then
                increase good(r)
            else if r miscorrects the classification of i-th segment in text' then
                increase bad(r)
            end if
        end for
    end for
    text := text'
end loop
```

OpenMPI [14] library was used, which also gives the possibility to run the tagger on multiple machines in parallel in a standardized way.

The workload is split between processes by distributing the transformation templates considered above (and the values of the **O** template variable, see section 3.2) among them. Therefore every process stores only a part of all the transformations. In each round of the algorithm the best transformation is collectively found, broadcast to all the processes, and the processing continues as shown in Algorithm 1.

4 Evaluation

The tagger was evaluated on two corpora of Polish: the IPI PAN Corpus of Polish [6] and the new National Corpus of Polish [15] (in preparation; version dated 2009–12–16 was used). The former corpus is allowed to have multiple golden tags for one segment, whereas the latter is fully disambiguated. For evaluation the manually disambiguated subcorpora were used, of size 880 000 and 648 000 segments, respectively.

The methodology proposed in [3] was used (which was also employed in [16]). A corpus was split into training part and evaluation part by ratio 9:1. The training part was used both as the training and patch corpus of the Brill algorithm. All taggers were configured to choose exactly one tag for each segment. Ten-fold cross-validated results are presented in Tables 2 and 3.

Table 2. Evaluation results — IPI PAN Corpus. Sources: [3, 16]

Tagger	Full tags					PoS only				
	C	WC	P	R	F	C	WC	P	R	F
Trigram HMM [17]	87.39	90.59	84.51	83.09	83.80	96.79	97.11	96.75	96.78	96.77
TaKIPI [2]	88.68	91.06	90.94	83.78	87.21	96.53	96.54	96.58	96.71	96.65
Brill [9] (2009)			89.46	83.55	86.40					
Brill (this paper)	90.00	92.44	92.44	86.05	89.13	98.17	98.18	98.18	98.16	98.17

Correctness (C) — percent of segments for which the tagger assigned exactly the same set of tags as the golden standard. Please note that in the IPI PAN corpus for some segments several tags are marked correct.

Weak correctness (WC) — percent of segments for which the sets of interpretations determined by the tagger and the golden standard are not disjoint.

Precision (P) — percent of tags (given by the morphological analyzer) which were both chosen by the tagger and the golden standard.

Recall (R) — percent of golden tags which were chosen by the tagger.

F-measure (F) — $F = \frac{2PR}{P+R}$.

Table 3. Evaluation results — National Corpus of Polish (full tags).

Thr[a]	Time (s)	# transformations		*Acc.* (%)
		P1	P2	
2	1450	5175.1	2748.0	92.82%
6	632	1422.6	612.2	92.68%

[a] The minimum $good(r) - bad(r)$ of the generated transformations.

Accuracy (*Acc.*) — any of the above in the case of the National Corpus of Polish, which has always one golden tag per segment.

The times in Table 3 were obtained on a multiprocessor machine with Xeon CPUs clocked at 2.4 GHz (with 12MB cache). 6 processes were run. The tagger was compiled in 64-bit mode, which probably negatively impacted performance due to almost doubled memory usage (∼1.2 GB per process compared to ∼0.7 GB in similar 32-bit setup), but this was not verified.

It is also worth noting that only TaKIPI does not disambiguate words not known to the morphological analyser, even if the input contains a number of possible morphosyntactic interpretations.

To provide a better insight into the classes of errors generated by the tagger, detailed statistics are presented in Tables 4, 5 and 6. It can be clearly seen that the most problems the tagger has concern CASE and GENDER. Slightly fewer errors are reported for NUMBER. This is similar to previous findings and not unexpected for Polish. Nevertheless, the introduction of lexical elements in transformation templates gave over 1.5% improvement in accuracy (on the National Corpus of Polish). Over 60% of all generated transformations do contain lexical matchers. The vast majority of them is used for determining the correct CASE by matching nearby segments' suffixes (see Table 7). Also, they are used for disambiguating rare flexemic classes like QUB from CONJ.

There are also some categories of errors in the testing corpush, which would not be disambiguated by a human looking at the same amount of context. Let us present several examples:

— Long nominal phrases, especially near sentence or subordinate clause boundaries:
`... tego znaku. Zamiłowanie do sportu i ...`
`... this sign. Passion for sport and ...`
Here the underlined word can have either nominal or accusative case.
— Expressions with words like *państwo*, which may be either a noun (*country*) or a pronoun (formal plural *you*):
`..., o czym państwo w tej chwili ...`
`..., what you/the country at the moment ...`

This calls for enlarging the lookup context in the future. For example, predicates like "the nearest segment with part-of-speech **P** has category **C** equal **X**" may be

Table 4. Error rates for parts of speech (shown only values > 0.01%).

Expected PoS	# errs	% toks
`subst`	3028	0.47%
`qub`	1658	0.26%
`adj`	1596	0.25%
`ger`	1392	0.21%
`conj`	652	0.10%
`adv`	597	0.09%
`ppas`	522	0.08%

Expected PoS	# errs	% toks
`prep`	471	0.07%
`pred`	363	0.06%
`num`	326	0.05%
`fin`	268	0.04%
`pact`	208	0.03%
`comp`	172	0.03%

Table 5. Error rates for grammatical categories (shown only values > 0.01%).

Category	# errs	% toks
CASE	21259	3.28%
GENDER	16151	2.49%
NUMBER	4645	0.72%
ASPECT	416	0.06%
ACCOMMODABILITY	193	0.03%

Table 6. Specific errors in assignment of grammatical categories (top 15 records).

Expected	Actual	# errs	% toks
CASE(NOM)	CASE(ACC)	7188	1.11%
CASE(ACC)	CASE(NOM)	4717	0.73%
GENDER(M1)	GENDER(M3)	2543	0.39%
CASE(GEN)	CASE(ACC)	2533	0.39%
NUMBER(SG)	NUMBER(PL)	2460	0.38%
NUMBER(PL)	NUMBER(SG)	2185	0.34%
GENDER(M3)	GENDER(M1)	1989	0.31%
GENDER(M3)	GENDER(N)	1662	0.26%
GENDER(M1)	GENDER(F)	1375	0.21%
GENDER(F)	GENDER(M3)	1243	0.19%
GENDER(M3)	GENDER(F)	1214	0.19%
GENDER(F)	GENDER(N)	1115	0.17%
CASE(GEN)	CASE(NOM)	1105	0.17%
CASE(ACC)	CASE(GEN)	963	0.15%
GENDER(N)	GENDER(M3)	907	0.14%

Table 7. Sample lexical transformations generated by the tagger in the first pass.

No.	r	**good**(r)	**bad**(r)
3	Change CASE of preposition from ACC to LOC if it ends with `na` (in practice this asks for the particular preposition *na*, in English: *on*) and one of two following segments has CASE of LOC.	2496	113
7	Change CASE of an adjective from LOC to INST if one of the three following segments has CASE of INST and ends with `em`.	921	29

good candidates for inclusion. This requires extending the vicinity parameter, and therefore slows down the computations, but may result in better accuracy.

5 Conclusions and future work

The paper presents and evaluates a number of techniques designed to adapt Brill tagger for inflectional languages with large tagsets. Especially adding predicates and actions which allow matching or changing values of single grammatical categories, as well as adding lexical transformations, were the most valuable modifications of the original algorithm.

It is worth noting that the tagger does not need any linguistic knowledge provided, except the specification of tagsets and the information about the grammatical categories which should be disambiguated in consecutive phases. Rule templates are not designed for any specific language. Even if some transformation templates are not suitable for considered language, they may negatively impact only performance, but not accuracy.

As far as the quality of the new tagger is concerned, the reported numbers are at least 1.1% higher than for other existing taggers for Polish, although this should be independently verified. Also, it may be an interesting experiment to use the tagger for other languages, like Czech or Hungarian (maybe after inclusion of all lexical transformations proposed by Megyesi [13]). There are also some other places for improvement not explored yet, namely:

1. Experimenting with different simplified tagsets and more than 2 passes. Tufiş [11] proposes using an additional reduced tagset to collapse grammatical categories which are unambiguously recoverable from the lexicon. This reduces the transformation space, improving performance. Others suggest joining some parts of speech or values of grammatical categories which have similar grammatical functions in the first pass, to disambiguate them later. For example in an intermediate phase one would use the value NOM-OR-ACC for CASE,
2. Simply enlarging the context of transformation templates may be a good way to go,
3. Designing transformation templates which look for the nearest segment with a particular part of speech or value of some grammatical category may improve accuracy.

The full source code of the tagger is available under the terms of the GNU GPL v3 from its project page: `http://code.google.com/p/pantera-tagger/`.

Acknowledgments

I'd like to sincerely thank my academic advisor Prof. Adam Przepiórkowski for his valuable help, always inspiring talks and effective motivation.

The research is funded in 2007–2010 by a research and development grant from the Polish Ministry of Science and Higher Education.

References

1. Spoustová, D.: Combining Statistical and Rule-Based Approaches to Morphological Tagging of Czech Texts. The Prague Bulletin of Mathematical Linguistics (89) (2008) 23–40
2. Piasecki, M., Godlewski, G.: Effective Architecture of the Polish Tagger. In Sojka, P., Kopecek, I., Pala, K., eds.: TSD. Volume 4188 of Lecture Notes in Computer Science., Springer (2006) 213–220
3. Karwańska, D., Przepiórkowski, A.: On the Evaluation of Two Polish Taggers. In: Proceedings of the 2009 PALC Conference in Łódź (to appear), Frankfurt/M., Peter Lang (2009)
4. Brill, E.: A simple rule-based part of speech tagger. In: Proceedings of the Third Conference on Applied Natural Language Processing, Morristown, NJ, USA, Association for Computational Linguistics (1992) 152–155
5. Przepiórkowski, A., Woliński, M.: A Flexemic Tagset for Polish. In: Proceedings of *Morphological Processing of Slavic Languages*, EACL 2003. (2003)
6. Przepiórkowski, A.: The IPI PAN Corpus: Preliminary version. Institute of Computer Science, Polish Academy of Sciences, Warsaw (2004)
7. Jurafsky, D., Martin, J.H.: Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition. Second edn. Prentice Hall (February 2008)
8. Piasecki, M., Wardyński, A.: Multiclassifier Approach to Tagging of Polish. In: Proceedings of 1st International Symposium Advances in Artificial Intelligence and Applications, unknown (2006)
9. Acedański, S., Gołuchowski, K.: A Morphosyntactic Rule-Based Brill Tagger for Polish. In: Recent Advances in Intelligent Information Systems, Kraków, Poland, Academic Publishing House EXIT (June 2009) 67–76
10. Ngai, G., Florian, R.: Transformation-based learning in the fast lane. In: NAACL '01 on Language technologies, Morristown, NJ, USA, Association for Computational Linguistics (2001) 1–8
11. Tufis, D.: Tiered Tagging and Combined Language Models Classifiers. In: TSD '99: Proceedings of the Second International Workshop on Text, Speech and Dialogue, London, UK, Springer-Verlag (1999) 28–33
12. Brill, E.: Some advances in transformation-based part of speech tagging. In: AAAI '94: Proceedings of the Twelfth National Conference on Artificial Intelligence (vol. 1), Menlo Park, CA, USA, American Association for Artificial Intelligence (1994) 722–727
13. Megyesi, B.: Improving Brill's POS tagger for an agglutinative language. In: Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora. (1999) 275–284
14. Gabriel, E., et al.: Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation. In: Proceedings, 11th European PVM/MPI Users' Group Meeting, Budapest, Hungary (September 2004) 97–104
15. Adam Przepiórkowski, Rafał L. Górski, B.L.T., Łaziński, M.: Towards the National Corpus of Polish. In: Proceedings of the Sixth International Language Resources and Evaluation. (2008)
16. Acedański, S., Przepiórkowski, A.: Towards the Adequate Evaluation of Morphosyntactic Taggers. In: Proceedings of the 23rd International Conference on Computational Linguistics. (2010) To appear.
17. Łukasz Dębowski: Trigram morphosyntactic tagger for Polish. In: Intelligent Information Systems. Advances in Soft Computing, Springer (2004) 409–413