Annales Mathematicae et Informaticae
49 *(2018) pp. 141–166*
DOI: 10.33039/ami.2018.05.001
http://ami.uni-eszterhazy.hu

Benchmarking morphological analyzers for the Hungarian language

Gábor Szabó, László Kovács

University of Miskolc
szgabsz91@gmail.com
kovacs@iit.uni-miskolc.hu

Submitted July 17, 2017 — Accepted May 22, 2018

Abstract

In this paper we evaluate, compare and benchmark the four most widely used and most advanced morphological analyzers for the Hungarian language, namely Hunmorph-Ocamorph, Hunmorph-Foma, Humor and Hunspell. The main goal of the current research is to define objective metrics while comparing these tools. The novelty of this paper is the fact that the analyzers are compared based on their annotation token systems instead of their lemmatization features. The proposed metrics for the comparison are the following: how different their annotation token systems are, how many words are recognized by the different analyzers and how many words are there whose morphological structure is equivalent using a well-defined mapping among the annotation token systems. For each of these metrics, we define the concept of similarity and distance. For the evaluation we use a unique Hungarian corpus that we generated in an automated way from Hungarian free texts, as well as a novel automated token mapping generation algorithm. According to our experimental results, Hunmorph-Ocamorph gives the best results. Hunmorph-Foma is very close to it, but sometimes returns an invalid lemma. Humor is the third best analyzer, while Hunspell is far worse than the other three tools.

Keywords: natural language processing, grammar, morphology, lemmatization, inflection, morphological analyzers, Hungarian language

MSC: 68T50

1. Introduction

Morphological analysis is an important part of every natural language processing (NLP) application. In NLP, processing of sentences is usually performed in several layers, and each layer depends on the results of the layers below it. Upper layers work on the sentence level, for example syntax tries to determine the structure of these sentences. Knowing the part of speech tag and the affixes of each word in a sentence helps in finding its functional components like the subject, the object and the predicate. Usually the ultimate goal of NLP is to *understand* a text in an automated way, finding semantic connections and concepts in the text. The basic form of describing these relationships is the so-called triple generation, where each triple contains a predicate, a subject and an object.

Morphology corresponds to the bottom layer, it's the lowest level of NLP that determines the structure of individual words. It's a subdomain of grammar. According to the Merriam-Webster dictionary, morphology is *"a study and description of word formation (such as inflection, derivation, and compounding) in language"*, so it works with intraword components. On the other hand, grammar works with higher level components, where the words are atoms.

The inner components of words according to morphology are the morphemes that are the smallest morphological units in a word that have an associated meaning, information [1]. Morphemes have two main categories: the lemma and the affixes. The lemma is the root form of a word that holds the base meaning of it. In many languages the meaning of the root form is modified using different affixes that are appended, prepended or simply inserted into the word. The process of adding affixes to a word is called inflection, while the inverse of inflection is lemmatization.

In most languages, there are other word formation techniques besides inflection as well. Derivation is a special kind of inflection, where the meaning of the root word is changed more dramatically, usually changing the part-of-speech tag as well. For example, *kindness* is a noun, derived from the adjective *kind.* On the other hand, a non-derivational inflection is (*chair*, *chairs*), where both words are nouns, and the -*s* suffix denotes the plural form. A third technique for creating new words is compounding, where two separate words are combined. An example for this technique is *fingerprint.*

Besides the lemma that is the grammatically correct root form of the word, we can also talk about stems, that can be generated from inflected forms by dropping all the affixes. It can be seen that in some cases the lemma and the stem will differ: for example the English word *tried* has the stem of *tri* and the lemma of *try.*

There are different types of affixes: prefixes are prepended to the root form of the word (***in****correct*), while suffixes are appended (*fly****ing***). Infixes are characters that are inserted in the middle of the words. They are very rare in most languages, an example is the Latin verb *vincō* where the *n* denotes present tense. In English most inflection rules are suffixes, while in Hungarian there are a couple of cases where we can find prefix affixes as well.

According to their morphological features, different languages represent different complexity levels and thus fall into different language categories [4]. Inflective languages such as English have a fix set of possible affix types for each part-of-speech category. Isolating languages like Chinese and Vietnamese usually have words that are their own stems, without any affixes. Languages that have only a few affix types usually use auxiliary words and word position to encode grammatical information, which is often said to be the characteristics of analytic languages. In intraflective languages (Arabic, Hebrew), consonants express the meaning of words, while vowels add the grammatical meaning. Synthetic languages have three subcategories: polysynthetic languages, fusional languages and agglutinative languages. Polysynthetic languages, including some Native American languages contain complicated words that are equivalent to sentences of other languages. Fusional languages like Russian, Polish, Slovak and Czech, have morphemes that are not easily distinguishable and often multiple grammatical relations are fused into one affix. Finally, agglutinative languages such as Hungarian, Finnish and Turkish have many affix types and each word can contain a potentionally infinite number of affixes.

In computational linguistics, morphological analyzers and parsers can be used to determine the morphological structure of the words in the chosen target language. As agglutinative languages such as Hungarian are very challenging to handle due to their many affix types and complex inflection rules, the goal of our research is to solve the inflection and lemmatization problems for Hungarian in an automated way.

There already exist some morphological analyzers for the Hungarian language. The motivation for this paper is to decide which of these well-known analyzers is worth using. To answer this question, we examine the following aspects:

- What are the most popular and usable morphological analyzers for the Hungarian language?
- What kind of metrics can we create to quantify the similarities and differences among them?
- How can we compare them objectively to decide which one is worth using?

The goal of this paper is to give a novel survey and evaluation on the available analyzers for the Hungarian language.

The results of this research are unique as we didn't find any similar papers that would objectively compare and benchmark morphological analyzers for the Hungarian language based on their ability to determine the lemma and the affix list of the given words.

As a preparation step, we had to create a training corpus large enough to be usable for the evaluation process. The corpus generation process was fully automated. The words were later used as the input of the morphological analyzers: Hunmorph-Ocamorph, Hunmorph-Foma, Humor and Hunspell. These tools determined the lemma, the part of speech and affix types in the words. After analyzing the morpheme model systems of the four tools, we also had to create a mapping among them as they had many differences in their nature.

For the evaluation process we came up with unique metrics such as similarity and distance to determine *how far* each analyzer is from each other. This way we can formalize the subjective experiences.

The main research goals of our examinations are:

1. What are the main differences of the analyzer outputs and token systems?
2. How many words are recognized by each analyzer?
3. How many words are there whose morphological structure is determined equivalently among the different analyzers?
4. What are the cumulative distances among the analyzers using the above three metrics?

The ongoing research topic where we'd like to use the results of this paper is the designing of a novel model and implementing a system built on this model that can solve the inflection, stemming and lemmatization problems. The structure of such a system can be seen in Figure 1[1].

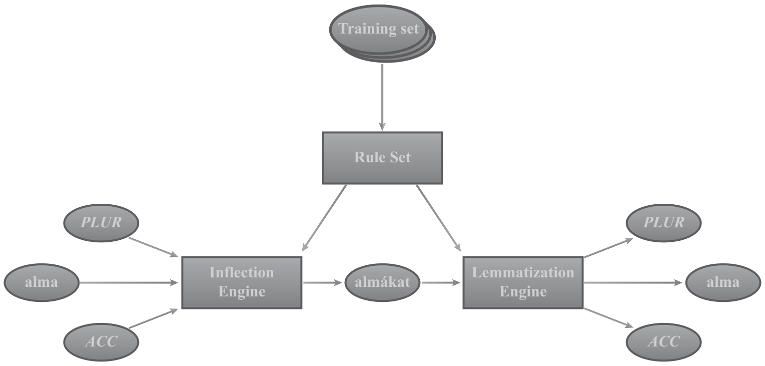

Figure 1: Automated inflection and lemmatization using generated morphological rules

In the figure we can see that the inflection and lemmatization engines use generated inflection rules that are derived from a training word pair set. This word pair set can be generated from a Hungarian corpus using a morphological analyzer: the analyzer determines the affix types in the words and using these morphological structures we can produce word pairs consisting of base forms and inflected forms. The applied analyzer will be the one that is proved to be the best by the results of this paper.

The structure of this paper goes as follows:

- Section 2 introduces the four most widely used morphological analyzers that will be evaluated.

[1]The word *almákat* is the plural form of *alma* (*apple*) in accusative case.

- In section 3 we can read about the motivation behind our experiments and the methodology of our research, introducing some formal theory behind the metrics we use.
- Section 4 presents the method of generating the training corpus that will be used as the input of the analyzers.
- The automated process of token mapping generation is summarized by section 5.
- The evaluation of the automated token mapping generation and the results of the manual token mapping can be seen in section 6 and the tables of the appendix[2].
- Finally, in section 7 we present analytical comparisons, different statistics and benchmarks on the examined analyzers.

2. Morphological analyzers

In literature, there are only a few existing publications that compare different morphological analyzers for the Hungarian language. The main comparison point of the papers we found is the stemming quality of these systems. They use test corpora containing words and calculate an error value for every record using overstemming and understemming indices [2].

We also found a very detailed Hungarian paper comparing some of the analyzers that we will evaluate, namely Hunmorph-Ocamorph [16], Hunmorph-Foma [5], Humor [13] and Hunspell [9]. The authors also brought in some other models that we didn't consider, as they can only be used for stemming purposes and not lemmatization or deeper morphological analysis: the Porter stemmer [10], the Hungarian adaptation of Snowball [11, 14] and some Apache Lucene [6] modules like KStem, Porter, EnglishMinimal, Stempel and Morfologik. The paper used metrics such as the number of words that the analyzers could process, how good the first answer was for each stemmer and how usable the other possible answers were compared to each other [3].

It's important to note that the goal of these papers was to evaluate the stemmer aspects of the tools, while in this paper we'd like to take care of lemmatization as well, which is a more complex problem, since stemming usually applies simpler rules that cut the typical word-ending characters, generating the stem that is not always the morphological root form. Lemmatization on the other hand always outputs a grammatically correct root word. That's why unlike the referenced papers that compare the resulting stems, we'll compare the morphological structure provided by the tools, i.e. the different affix types that appear in the output for each word. We'll compare these lists and provide different statistics about them, like the total

[2] `http://users.iit.uni-miskolc.hu/~szabo84/articles/morphological-analyzers/appendix.pdf`

number of words that are recognized by the analyzers, the number of recognized words, and the number of words whose structure is the same across the multiple outputs.

Similar tools exist for other languages as well. For example morpha, morphg [8] and ana [7] are three morphological analyzers for the English language. The first two can solve the stemming and inflection problems using finite state machines, while the third one can insert the appropriate indefinite articles (*a* or *an*) into free texts. However, our target language is the Hungarian language, therefore we chose the four possible analyzers for Hungarian language:

1. Hunmorph-Ocamorph [16]
2. Hunmorph-Foma [5]
3. Humor [13]
4. Hunspell [9]

In the following subsections we will introduce them one by one.

2.1. Hunmorph-Ocamorph

Hunmorph-Ocamorph [16] is the first morphological analyzer that we examine. It is developed by the Budapest University of Technology and Economics, and is based on the *ocamorph* morphological analyzer tool, using the Morphdb.hu [15] Hungarian lexical database and morphological grammar. Both the tool and its engine are fully open-source.

The analyzer engine was designed in a way that is totally language independent so that it can work with dictionaries of any language. The dictionary for the Hungarian language is called Morphdb.hu, and stores lexical records with different affix flags. The analyzer uses these pieces of information to generate the output, which is the lemma and the list of affix tokens contained by the database. Usually this dictionary is in a very compact format, not allowing linguistics to directly build it, that's why Hunlex was developed, that can generate the *dic* and *aff* files accordingly from central lexical databases.

To preserve disk and memory space, the dictionary records do not contain every possible inflected form of the root lemma, instead the possible forms are described using general rules and the unpredictable irregular forms. This means that most of the inflected forms can be covered with a small amount of rules, and only the irregular word forms must be taken care of additionally.

The transformation rules in the database have two main categories: the first group of rules are morphosyntactically active meaning they add something (a morpheme) to the root, while filter rules modify the previously existing form, changing vowel length or other aspects of it. Each of these rules are connected to different features of the words. If the associated features of a rule can be found in the input word, then the transformation rule is applied. Note that the rules can be used both for inflection and stemming, because these operations are the inverse of each other.

2.2. Hunmorph-Foma

Hunmorph-Foma [5] is very similar to the previous morphological analyzer, but it is based on the *foma* analyzer engine and has different annotation tokens.

Foma is *"a compiler, programming language, and C library for constructing finite-state automata and transducers for various uses"* [5] that is the open-source implementation of the *lexc/xfst* grammatical analyzer of the Xerox laboratory.

Hunmorph-Foma is also open-source, and has an introductory readme file on its Github page[3]. According to that, Hunmorph-Foma is better than Hunmorph-Ocamorph as it is based on a broader corpus, has lower memory consumption and is overall faster. The tool is also compared to Hunspell, and according to the documentation, Hunspell is far behind both Hunmorph based analyzers.

In section 7 we'll see how these analyzers can be compared to each other and what the results will be like for our case.

2.3. Humor

Humor [13, 12] (High-speed Unification MORphology) is a closed-source morphological analyzer implementation developed by MorphoLogic[4]. Unlike the other examined tools, this one is a commercial product with closed-source, only the DLL is present, the internal implementation details are not visible to us.

Humor is a rule based system, similarly to the previous two tools. Rule based design is well suited for highly agglutinative languages like Hungarian as in this language one word might have many many inflected forms, in some extreme cases almost a thousand different word forms and variations, so storing all of them in a dictionary would require much disk space and memory.

Humor specifically targets the Hungarian language. During the processing of an input word, the analyzer engine breaks up the word to morphs that are the concrete appearances of morphemes, validates the connections of neighboring morphs and checks the validity of the full morphological context. This means that it not only determines the morphological structure of the words, it also checks if the possible affix candidates can be in each other's context. The neighborhood validation is done using morpheme features, similarly to other morphological analyzers. If the tool finds that two neighboring morphemes have mutually exclusive features, then the morphological structure is marked invalid and dropped.

The added value of Humor over the Hunmorph based analyzers is that it not only determines the affix types, it also shows which part of the word is associated with which affix. If the lexical and surface form of the morpheme differs, the tool returns both of them. This happens sometimes when the base form of a word changes according to the added affix.

For each input word, Humor might give back multiple morphological structures, among which some can look unfamiliar or strange. This is because they are valid according to the rules, but are rarely or never used. Therefore Humor tries to

[3]`https://github.com/r0ller/hunmorph-foma`

[4]`https://www.morphologic.hu`

return the more accepted forms first, and leave the weird forms to the end of the list. This is done using occurrence counters.

2.4. Hunspell

Hunspell [9] is a popular open-source spell checker of LibreOffice, OpenOffice.org, Mozilla Firefox and Thunderbird, Google Chrome, etc. Although it's mainly used for spell checking, it has other use cases as well, including morphological analysis.

Similarly to Hunmorph, the language database consists of two files: a dictionary file containing lemmas and an affix file with the possible affix tokens, features and transformation rules.

If the authors of Hunmorph-Foma are right, we will experience that Hunspell is less usable than the Hunmorph based analyzers.

3. Methodology

In this section we summarize the steps we'll take in the comparison of the four examined morphological analyzers and introduce the formal model of the benchmarking process.

First, let's have a corpus W containing a set of valid words for the target language: $W = \{w_1, w_2, \ldots, w_k\}$. We denote the set of lemmas in the language with $\bar{W} \subset W$. The automated process of corpus generation is described in section 4.

$\mathbb{A} = \{A_1, A_2, \ldots, A_n\}$ denotes the set of investigated morphological analyzers. Each analyzer has an associated set of annotation tokens: analyzer A_i is associated with the token set $T_i = \{t_{i_1}, t_{i_2}, \ldots, t_{i_l}\}$. We assume that for every morphological analyzer pair $(A_i, A_j), i \neq j$, the corresponding token sets are disjoint: $T_i \cap T_j = \emptyset$. If only one analyzer is examined in the next sections, we denote it with A, omitting its index. If an analyzer pair is examined, they will be denoted by (A_1, A_2), and so on.

The morphological analyzer A_i maps every word to a lemma and an ordered list of tokens denoting the affix types, or an error state denoted by ϵ meaning that the input word wasn't recognized by A_i:

$$A_i : W \to \left(\bar{W} \times \langle t_{i_{j_1}}, t_{i_{j_2}}, \ldots, t_{i_{j_m}} \rangle \cup \epsilon\right)$$

The projection for the lemma part is denoted by $\lambda\left(A_i\left(w\right)\right)$, while the projection for the token list part is denoted by $\tau\left(A_i\left(w\right)\right)$.

For a given morphological analyzer A_i, the set of recognized words is denoted by

$$W_i = \{w \in W \mid A_i\left(w\right) \neq \epsilon\}$$

We can also determine the ratio of W_i compared to the full corpus:

$$\mu_i = \frac{|W_i|}{|W|}$$

Based on the number of recognized words we can define the recognition similarity of two morphological analyzers A_i and A_j as

$$\sigma^R_{i,j} = \frac{|W_i \cap W_j|}{|W_i \cup W_j|} \tag{3.1}$$

We can also define the recognition distance metric based on the recognition similarity:

$$\delta^R_{i,j} = \frac{1}{\sigma^R_{i,j}} \tag{3.2}$$

The results for the number of recognized words, recognition ratio, recognition similarity and distance can be seen in subsection 7.2.

For any morphological analyzer pair (A_i, A_j), the set of comparable words are those words whose lemma is the same according to both analyzers:

$$W_{i,j} = \{w \in W \mid \lambda(A_i(w)) = \lambda(A_j(w))\}$$

As the annotation token systems of the morphological analyzers may differ significantly, we define a mapping among them as $\chi_{i,j} \subseteq T_i \times T_j$. Using this mapping, we can easily map a token list:

$$\chi_{i,j}(\langle t_{i_1}, t_{i_2}, \ldots, t_{i_o}\rangle) = \langle \chi_{i,j}(t_{i_1}), \chi_{i,j}(t_{i_2}), \ldots, \chi_{i,j}(t_{i_o})\rangle$$

Similarly, a set of tokens can be mapped easily as well:

$$\chi_{i,j}(\{t_{i_1}, t_{i_2}, \ldots, t_{i_o}\}) = \{\chi_{i,j}(t_{i_1}), \chi_{i,j}(t_{i_2}), \ldots, \chi_{i,j}(t_{i_o})\}$$

The given mapping is perfect if

$$\forall w \in W_{i,j} : \chi_{i,j}(\tau(A_i(w))) = \tau(A_j(w))$$

Usually we cannot define a perfect mapping because there might be affix types that are recognized by A_i but not recognized by A_j.

Section 5 will present a novel automated token mapping algorithm, while section 6 will evaluate this automated process and compare its results to a manual mapping introduced in the tables of the appendix[5].

Using the mapping, we can compare the recognized tokens of the morphological analyzer by calculating the token similarity metric:

$$\sigma^T_{i,j,\chi_{i,j}} = \frac{|\chi_{i,j}(T_i) \cap T_j|}{|\chi_{i,j}(T_i) \cup T_j|} \tag{3.3}$$

[5]`http://users.iit.uni-miskolc.hu/~szabo84/articles/morphological-analyzers/appendix.pdf`

The token distance between two analyzers can be calculated with the following formula:

$$\delta^{T}_{i,j,\chi_{i,j}} = \frac{1}{\sigma^{T}_{i,j,\chi_{i,j}}} \tag{3.4}$$

In subsection 7.1 we'll introduce the main differences among the annotation token systems, as well as the exact numbers of token similarity and distance.

To measure how many words are there whose morphological structures are equivalent according to two morphological analyzers using a given mapping, we can first calculate the equivalence similarity as

$$\sigma^{E}_{i,j,\chi_{i,j}}\left(A_i, A_j\right) = \frac{\left|\left\{w \in W \mid \chi_{i,j}\left(\tau\left(A_i\left(w\right)\right)\right) = \tau\left(A_j\left(w\right)\right)\right\}\right|}{\left|W_i \cup W_j\right|} \tag{3.5}$$

Using the equivalence similarity, we can calculate the equivalence distance:

$$\delta^{E}_{i,j,\chi_{i,j}}\left(A_i, A_j\right) = \frac{1}{\sigma^{E}_{i,j,\chi_{i,j}}\left(A_i, A_j\right)} \tag{3.6}$$

Subsection 7.3 will introduce the results for the equivalence similarity and equivalence distance.

Using the above distance definitions, we can calculate a cumulative distance among the analyzers using the following formula:

$$\delta^{C}_{i,j,\chi_{i,j}}\left(A_i, A_j\right) = \delta^{R}_{i,j} + \delta^{T}_{i,j,\chi_{i,j}} + \delta^{E}_{i,j,\chi_{i,j}} \tag{3.7}$$

This cumulative distance will be detailed by subsection 7.4. For all the distances, we will also plot simple 2D Cartesian coordinate systems to visualize our measurements.

4. Building the test corpus

The first step towards comparing multiple morphological analyzers is to create an appropriately big test corpus, gathering a word set large enough to be usable. Having such a word set we can ask each tool what kind of morphological structure these words have and then compare the results.

The Hungarian Electronic Library[6] was chosen as the source of the words. This is a huge site containing many documents (16 250 at the time of writing) in Word, PDF, HTML, etc. formats. Some documents are stored in multiple formats, while others have only one stored version. The site has a paginated full list of these documents that was used as the starting point. When we click on the title of a chosen record, a details page is loaded with links to the available formats.

To speed up the process of word extraction, a fully automated method was implemented in the Java language. The programming language was chosen because of

[6]`http://mek.oszk.hu`

its functional features introduced in Java 8 and the advantages of parallel streams. The pseudocode of the algorithm can be seen in Listing 1.

Listing 1: Word Extraction Algorithm (Pseudocode)

```
getPages()
    .parallelStream()
    .flatMap(page -> getDocumentDetailsStreamOn(page))
    .map(documentDetail -> chooseAFormat(documentDetail))
    .map(document -> parseToString(document))
    .flatMap(content -> splitToWordStream(content))
    .forEach(word -> consumeWord(word));
```

We iterate all the pages in parallel, using parallel streams that are built on top of the Java 7 Fork/Join framework. The internal implementation of this framework knows how many CPU cores we have and thus splits up the work into that many chunks that are executed in parallel. Each page is flattened out to the stream of document details on that page. These details are also processed in parallel: first, we choose a format from the available ones, then we read that document into a string, split that into a list of words, and finally we process the words and store them in a CSV file containing the words and the number of their occurrences. For the document processing, open source Java libraries were used like Apache POI[7] for Word, iText[8] for PDF and jsoup[9] for HTML.

After executing this automated algorithm, the total number of lines in the final CSV file was 13 345 903. It's important to note that this is not the total number of meaningful words because of some bugs in the document processing libraries, typos in the documents, etc. These are just word candidates that either will be proven to be real words or will be dropped in the next phase. The exact numbers of the recognized words can be seen in subsection 7.2.

5. Automated algorithm for token mapping generation

The comparison of different morphological analyzers based on their annotation token systems and the ability of determining the morphological structure of input words requires a well-defined mapping among their token systems. There is two possible ways of creating such a mapping: creating an automated algorithm for constructing the mapping based on a training data, or using a manual process for doing the same thing. The automated algorithm is the only solution in general, but for our case the manual mapping is possible as well.

During our evaluation process, we tried both routes: first, we created an automated algorithm for generating a token mapping based on relative probabilities,

[7]`https://poi.apache.org/`
[8]`http://itextpdf.com/`
[9]`https://jsoup.org/`

then examined the results and fixed the errors we found in the statistical mapping. In this section we describe the automated algorithm and its theoretical background, then in section 6 we compare the results of the automated and the manual process.

Let's have two morphological analyzers A_i and A_j, a word w and the two corresponding results containing the lemmas $\lambda\left(A_i\left(w\right)\right)$, $\lambda\left(A_j\left(w\right)\right)$ as well as tokens $\tau\left(A_i\left(w\right)\right)$, $\tau\left(A_j\left(w\right)\right)$.

To build the appropriate mapping between $\left(A_i, A_j\right)$, we use a training set of $\left\{\left(A_i\left(w\right), A_j\left(w\right)\right) \mid w \in W \ \wedge \ \lambda\left(A_i\left(w\right)\right) = \lambda\left(A_j\left(w\right)\right)\right\}$. The lemmas need to be the same for the corresponding items, because otherwise we would also map token lists related to completely different words. Regarding the two different token systems we assume that they are disjoint: $T_i \cap T_j = \emptyset$.

The mapping can be formalized as a relation of $\chi_{i,j} \subseteq \mathcal{P}\left(T_i\right) \times \mathcal{P}\left(T_j\right)$, because in practice there can be cases when either the left side or the right side doesn't exist for a mapping.

One condition for the mapping is that each token can only appear at most once both on the left side (the domain) and the right side (the range):

$$\forall x \in \mathit{Dom}\left(\chi_{i,j}\right) : \left|x \ \chi_{i,j}\right| = 1$$
$$\forall y \in \mathit{Ran}\left(\chi_{i,j}\right) : \left|\chi_{i,j} \ y\right| = 1$$

For simplicity, the notation $\chi_{i,j}\left(x\right)$ can be used in place of both $x \ \chi_{i,j}$ and $\chi_{i,j} \ x$.

As the different annotation token systems can be very different, both the domain and the range of the mapping relation contain token lists of any length. For simpler cases, one token of A_i can be mapped to another token of A_j, but in more complex cases one token of A_i is covered by a token list of A_j or vice versa.

A naïve mapping between the annotation token systems of two morphological analyzers A_i and A_j would be

$$x \ \chi_{i,j} \ y \Leftrightarrow \exists w \in W : \left(A_i\left(w\right) = \left(w_0, x\right) \ \wedge \ A_j\left(w\right) = \left(w_0, y\right)\right)$$

However, this mapping would have many rules and wouldn't be able to generalize. That's why we also want to optimize two metrics of the mapping relation:

$$\left|\chi_{i,j}\right| \rightarrow min!$$
$$\sigma^E_{i,j,\chi_{i,j}}\left(A_i, A_j\right) \rightarrow max!$$

Based on these goals the automated process must be able to generate a mapping relation for all the examined morphological analyzers: Hunmorph-Ocamorph, Hunmorph-Foma, Humor and Hunspell. For simplicity, we create a mapping between Hunmorph-Ocamorph, and all the other analyzers, treating the token system of Hunmorph-Ocamorph as the base. The steps of the algorithm for any two analyzers are the following:

1. Generate a set of candidate mapping items based on relative probability within the training data set.
2. $\forall x \in Dom\left(\chi_{i,j}\right)$ choose $y \in Ran\left(\chi_{i,j}\right)$ that has the highest probability. Do the same for the remaining token sequences of the range.
3. Eliminate all the redundant pairs $(x, y) \in \chi_{i,j}$ where $\exists\left(x', y'\right) \in \chi_{i,j}$ and $\exists\left(x'', y''\right) \in \chi_{i,j}$ such that $x = x' + x''$ and $y = y' + y''$.

Algorithm 1 contains the pseudocode for this automated process.

Algorithm 1 Generating a statistical mapping

```
1: procedure GenerateStatisticalMapping(records1, records2)
2:     $r \leftarrow$ EmptyMap ()
3:     $m \leftarrow$ EmptyMap ()
4:     for all $(r_1, r_2) \in$ GeneratePairs $(records1, records2)$ do
5:         $C \leftarrow$ GenerateCombinations $(r_1, r_2)$
6:         for all $c \in C$ do
7:             $v \leftarrow 0$
8:             if $m[c]$ then
9:                 $v \leftarrow m[c]$
10:            end if
11:            Put $(m, v + 1)$
12:        end for
13:    end for
14:    RemoveNonAtomicEntries $(m)$
15:    SortEntries $(m)$
16:    for all $e \in m$ do
17:        Put $(r, e)$
18:        RemoveSimilarEntries $(m, e)$
19:    end for
20:    return $r$
21: end procedure
```

The input of the procedure is the resulting (lemma, token list) pairs of two analyzers. The *GeneratePairs* subprocedure generates pairs from the given lists, where every pair is related to the same input word and has the same lemma. *GenerateCombinations* returns a set of token list pairs where all the possible token sequences appear. For example a token pair *(abc, ef)* will yield the following combinations: *(a, e)*, *(bc, f)*, *(ab, e)* and *(c, f)*. From all these possible combinations, we count the occurrences of all the possible mapped token lists. After that, we remove all the entries that can be produced by combining other shorter token lists. For example if we have the candidates *(abcd, efg)*, *(ab, e)*, *(c, f)*, *(d, g)*, then we can see that the first candidate can be removed as it is the combination the latter three. The remaining entries are sorted by their relative probabilities (frequencies), in descending order. The resulting mapping is generated from the most frequent entries, but we also make sure that if an item is put in the resulting map, we remove all the other entries that have the same left or right side.

To improve the results, we also applied a second phase after this automated algorithm, where we did not generate all the possible pairs (line 4), but only those whose probability was the highest based on the frequency map of the previous phase.

6. Evaluation of the automated mapping construction algorithm

Before the evaluation, let's see how the manual mapping is structured. We grouped the different tokens by their categories: part-of-speech categories, noun and verb features, different derivation types and postposition subcategories. The mapping for the different categories are summarized by the tables of the appendix[10]as the tables would be too big to include them in this section. The tables contain the tokens of the four analyzers, as well as the name of the affix and a word or phrase in Hungarian and English. However, sometimes it was hard to give a perfect English translation of the Hungarian word, while in other cases the same word is used for different affix types. This could be remedied using whole sentences, but it would require more space, so we sticked with only words and phrases.

In this section we only provide some explanation for the different tables.

First, let's examine the part-of-speech category tokens in Table 1 of the appendix. As we can see, most POS categories exist in all models, but some rare ones only exist in Hunmorph-Ocamorph. This is not a big problem, as the common POS categories are more important for our case.

Table 2 of the appendix contains all the tokens for noun features.

As we can see, Ocamorph and Foma can describe all features, while some are missing from Humor and Hunspell. Some tokens in the column of Hunspell are marked with an asterisk, which means that the actual token is longer, but it contains the prefix appearing in the cell, which identifies the whole token uniquely. This way we saved some space. An example is the token *ék** which is in reality *ék_FAMILIAR_noun*.

We can also see the first few differences in the token range among the four token systems: Foma, Humor and Hunspell have more fine-grained tokens than Ocamorph. For example, the Foma token *Possp3p* is associated with two tokens in Ocamorph: ⟨*PLUR*⟩ and ⟨*POSS*⟨*PLUR*⟩⟩. This will be observable in other tables as well.

Table 3 of the appendix contains the tokens for verb features. As all the samples use the Hungarian word *fut (run)*, the English counterparts are not provided.

[10]`http://users.iit.uni-miskolc.hu/~szabo84/articles/morphological-analyzers/appendix.pdf`

Instead, the features appearing in the inflected form of each row can be deduced from the first column's description.

Here, we can find similar rows to the noun features, where Ocamorph requires multiple tokens to describe the same features as only one token in other columns, for example the Foma token *Inf11* is represented with two tokens in Ocamorph, namely ⟨*INF*⟩ and ⟨*PERS*⟨*1*⟩⟩. Also, Humor lacks multiple affix tokens compared to the other analyzers.

Note that the token ⟨*VPLUR*⟩ does not exist in the database of Hunmorph-Ocamorph, but we had to introduce it to avoid ambiguity. This is because the 3rd person plural indefinite form of verbs have the ⟨*PLUR*⟩ token by default, just like plural nouns, which would lead to errors by the time we start generating word pairs: the same affix type would have two kinds of inflection. That's why we decided to modify the original token to make sure that the two inflections have different tokens and thus can be easily distinguished.

As for derivations, it seems that Ocamorph is superior, because Foma, Humor and Hunspell lack many of the token types. Table 4 of the appendix shows all the derivation types where the source POS category is */NOUN*, table 5 of the appendix contains the tokens for derivations whose source POS category is */VERB*, table 6 of the appendix contains the tokens for derivations whose source POS category is */ADJ*, while table 7 of the appendix shows the tokens for derivations whose source POS category is */NUM*.

It's interesting that in some cases, one token in Humor means multiple things. For example, *[FF]* means both *[SUPERLAT]* and *[SUPERSUPERLAT]*. However, as this case is ambiguous, we only included it in one row.

Table 8 of the appendix shows all the different postposition subcategories. Most of the tokens here are only recognized by Hunmorph-Ocamorph, but for the sake of completeness we include these tokens as well.

All in all, we can state that the two main conclusions from the token mapping tables are that in some cases Ocamorph describes affix types using multiple tokens while the other token systems have more fine-grained token types, as well as Foma, Humor and Hunspell have multiple missing tokens, especially in the area of derivations.

With this manual mapping, we got a clear picture about all the relations between the different tokens. However, creating the mapping manually was a very time-consuming and error-prone process, as we had to examine many of the analyzer results by hand. The automated mapping generation has the advantage of being much faster, even though it can produce invalid mappings as well.

First, let's see the runtime performance of the automated algorithm in Table 1.

As we can see, the algorithm finished in around 2.5 minutes even for the slowest case. The differences in runtime can come from the differences of the token systems and the number of results with the same lemma.

Although the runtime of the algorithm was superior compared to the manual process, we also found that there were some invalid mappings in the results. The reason is that for some words the analyzers returned multiple ambiguous morpho-

A_1	A_2	**Runtime [seconds]**
Ocamorph	Foma	108.83
Ocamorph	Humor	38.37
Ocamorph	Hunspell	152.76

Table 1: Runtime of the automated statistical mapping generation algorithm

logical structures with the same lemma. An example is the word *olvastam (I read (past tense))* where Hunmorph-Ocamorph returned two results:

- *olvas/VERB⟨PAST⟩⟨PERS⟨1⟩⟩⟨DEF⟩* and
- *olvas/VERB⟨PAST⟩⟨PERS⟨1⟩⟩*.

The difference is that the first morphological structure is definite while the second one is indefinite, because the analyzer cannot decide which one is the correct result for the given inflected form. This is a linguistic problem that cannot be eliminated in an automated way and can be observed with all the examined analyzers, so these ambiguous results led to some invalid mapping items like

- *CondDefPl1* mapping to *⟨COND⟩⟨PERS⟨1⟩⟩⟨PLUR⟩* instead of
- *CondDefPl1* mapping to *⟨COND⟩⟨PERS⟨1⟩⟩⟨PLUR⟩⟨DEF⟩*.

Table 2 contains the percent values for the statistical mapping correctness compared to the manual process.

A_1	A_2	**Mapping Correctness**
Ocamorph	Foma	85.71%
Ocamorph	Humor	73.73%
Ocamorph	Hunspell	73.01%

Table 2: Correctness of the statistical mapping

We can see that the generated mapping between Hunmorph-Ocamorph and Foma proved to be 85.71% correct, while the mappings between Ocamorph and Humor, as well as Ocamorph and Hunspell became a bit worse. This can be due to the conceptual differences of these morphological analyzers.

7. Analytical results

In this section, we present the analytical results for the metrics mentioned in section 3: the differences among the token systems, the number of recognized words, the equivalence of the morphological structures provided by the analyzers and the cumulative distances. We also provide Cartesian coordinate systems to visualize the resulting distances in 2D space.

7.1. Comparison of the token systems

In this subsection, we highlight the main differences in the token systems of the examined analyzers.

Let's start the analysis from the end of the list: Hunspell. It is not surprising that among other morphological analyzers Hunspell prove to be not so strong, as it is rather a spell checker tool than a morphological analyzer. There are multiple missing tokens compared to Ocamorph, and the output results are more bloated as well.

Next in line is Humor. Although its source code is not visible for the end user (it is a closed-source product), the experience shows that this tool is more undeterministic than the others. There were cases when the same word caused runtime errors once and produced a valid result other times. This can be due to it being an older version, maybe newer versions of the product would perform better. Also, it seemed that the tool couldn't handle special accented vowels well, because there were many cases where words with such characters haven't been recognized even if the overall morphological structure of the word was rather simple.

A positive point about Humor is that it also recognized compound words. A good example is the word *szekrényajtót* (accusative case of *wardrobe door* in Hungarian). Other tools tend to only recognize that this is the accusative case of the word *szekrényajtó* (*wardrobe door*), while Humor also recognized that it is a compound word (*szekrény (wardrobe) + ajtó (door)*). However, for the current research, not being able to recognize compound words is not a real problem as we are only interested in affixes and inflection. In case of Humor we would have to merge the components of compound words to extract this information properly.

Another good aspect of Humor (and also Foma and Hunspell) is that its tokens are more fine-grained than those of Hunmorph-Ocamorph.

For the first sight, Hunmorph-Foma might seem the best tool out of the four. It has fine-grained tokens that help the word pair generation process, it has the highest total number of recognized words (see subsection 7.2), and so on.

However, during the analysis we found some cases where the stem is incorrect. A simple example is *(merengők, mereng+Adj+Plur)* which is the plural of *meditative* in Hungarian. The morphological structure says that the stem is *mereng* (*meditate*) which is in reality a verb, but according to the tokens it is an adjective. The correct structure would be *merengő+Adj+Plur*. If there are such incorrect records in the final CSV file, during the word pair generation process there will be incorrect word pairs that distort the future training set. Instead of the correct *(merengő, merengők)* we would generate *(mereng, merengők)* which is incorrect.

Another problem with this tool was the handling of adjective derivation. A sample record is *(legeslegszebbik, legeslegszép+Adj+Mid+Ik)* which is the supersuperlative designative form of the word *nice* in Hungarian. This structure suggests that the lemma is *legeslegszép* which is in reality not a meaningful Hungarian word, but rather a semi-inflected form of the real root form *szép* (*nice*), containing the supersuperlative prefix affix as well. Hunmorph-Ocamorph can recognize this fact and produce the right lemma.

Although these cases are not general as we only found a couple of them, these differences suggested us that Hunmorph-Ocamorph is slightly better than Hunmorph-Foma.

With the notations of section 3, specifically equations 3.3 and 3.4, we gathered the similarity and distance values among the examined morphological analyzer annotation token systems in Table 3.

A_1	A_2	$\sigma^T_{1,2,\chi_{1,2}}$	$\delta^T_{1,2,\chi_{1,2}}$
Ocamorph	Foma	0.5707	1.7521
Ocamorph	Humor	0.5171	1.9340
Ocamorph	Hunspell	0.6927	1.4437
Foma	Humor	0.4049	2.4699
Foma	Hunspell	0.5171	1.9340
Humor	Hunspell	0.4488	2.2283

Table 3: Similarity $\left(\sigma^T_{i,j,\chi_{i,j}}\right)$ and distance $\left(\delta^T_{i,j,\chi_{i,j}}\right)$ values among the annotation token systems

Figure 2 visualizes the approximation of these distance values to make them easier to imagine. It seems that Ocamorph and Hunspell have the closest token systems, however, this doesn't mean that both tools are advanced in other areas, as we'll see in the next subsections.

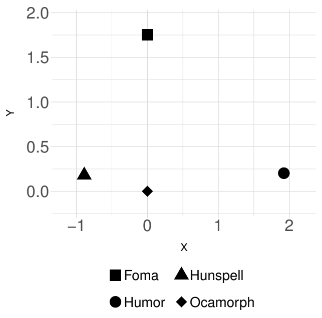

Figure 2: Visualizing the annotation token system distances

7.2. Recognition statistics

Let's investigate how many word candidates in the corpus have been recognized by the different morphological analyzers and in how many cases did the analyzers throw errors. As we discussed previously, a word w is recognized by analyzer A_i if $A_i(w) \neq \epsilon$.

With this definition in mind, it is a good starting point in the comparison process to determine how many words are common among the different analyzers. Table 4 contains the total number of results, the number of recognized words

($|W_i|$) and the ratio of recognized words compared to the whole corpus (μ_i). The denominator of μ_i, i.e. the total number of unique words recognized by at least one analyzer is 3 852 592. Since the input words might have multiple different morphological structures returned by the analyzers, the first number is always higher than $|W_i|$.

A	**Total Results**	$\lvert W\rvert$	μ
Hunmorph-Ocamorph	4 423 882	2 515 570	65.30%
Hunmorph-Foma	3 759 201	3 154 529	81.88%
Humor	1 017 004	823 531	21.38%
Hunspell	148 828	99 441	2.58%

Table 4: The number of total results, the number of recognized words ($|W_i|$) and the ratio of recognized words (μ_i) for each morphological analyzer

As we can see, Hunspell and Humor both recognized the least amount of words, so that points to the conclusion that these two analyzers are weaker than the Hunmorph based ones.

Let's look at the number of commonly recognized words among the analyzers: how many words are recognized by more of the examined analyzers. First, let's see the statistics of table 5 that show the number of words recognized by only a single analyzer.

A	**Words Recognized only by** A
Ocamorph	159 651
Foma	723 968
Humor	387 380
Hunspell	16 256

Table 5: The number of words only recognized by one analyzer

Hunspell has the lowest number, the second lowest one is for Ocamorph. This could be surprising, but we will see later that the cause of this is that Ocamorph's recognized words are usually recognized by Foma as well. On the other hand, Foma has the highest number in the table, which means that there are many words recognized only by Foma. These are mostly special cases like conjunction, postposition, etc.

Next, let's look at the number of words recognized by exactly two morphological analyzers in Table 6.

As we can see, the Hunmorph based tools have many commonly recognized words, while the others are far behind them.

Table 7 shows the commonly recognized words among exactly three analyzers.

This shows that if we include Hunspell, the number of items in the intersection of the three sets drops dramatically, and in this regards Humor is better than

	Ocamorph	**Foma**	**Humor**
Foma	1 997 989	-	-
Humor	12 132	69 705	-
Hunspell	748	16 262	7 610

Table 6: The number of words recognized by exactly two analyzers

Morphological Analyzers	**Recognized Words**
Ocamorph-Foma-Humor	336 339
Ocamorph-Foma-Hunspell	48 200
Ocamorph-Humor-Hunspell	99
Foma-Humor-Hunspell	1 654

Table 7: The number of words recognized by exactly three analyzers

Hunspell.

Finally, the number of words recognized by all the four morphological analyzers is 8 612.

To visualize the number of recognized words for the three strongest tools, namely Hunmorph-Ocamorph, Hunmorph-Foma and Humor, let's look at the Venn-diagram in Figure 3. We omitted Hunspell as it has far worse metrics, so that we can use a simple 2D visualization. Therefore the values of Figure 3 are a bit different than the previous tables, because the metrics of Hunspell are not included.

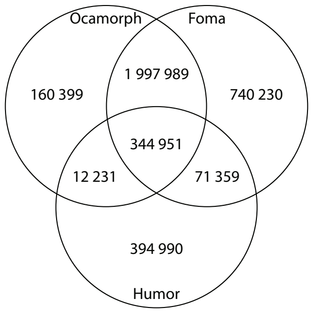

Figure 3: Venn diagram of the number of recognized words for the three strongest morphological analyzers

The Humor tool (as well as Hunspell whose data is not included in the diagram) is far worse than the Hunmorph-Ocamorph and Hunmorph-Foma for the first sight, as it could recognize far less words than the Hunmorph based tools, and the intersection sizes were also too low.

The conclusion, like in the previous subsection is that the Ocamorph and Foma based analyzers are the best, and they also have many recognized words in common.

The only major issue with Foma so far is that in some cases it handles stems incorrectly.

Table 8 shows the similarity and distance values for all the possible analyzer pairs based on equations 3.1 and 3.2.

A_1	A_2	$\sigma^R_{1,2}$	$\delta^R_{1,2}$
Ocamorph	Foma	0.4132	2.4201
Ocamorph	Humor	0.1070	9.3485
Ocamorph	Hunspell	0.0220	45.3530
Foma	Humor	0.0898	11.1373
Foma	Hunspell	0.0177	56.4347
Humor	Hunspell	0.0195	51.3475

Table 8: Similarity ($\sigma^R_{i,j}$) and distance ($\delta^R_{i,j}$) values among the morphological analyzers based on the number of recognized words

Figure 4 visualizes the approximation of these distance values. Hunmorph-Ocamorph and Hunmorph-Foma are very close to each other indeed, and Humor is not far from them either. On the other hand, Hunspell is far from the three other analyzers, as we saw in the previous tables.

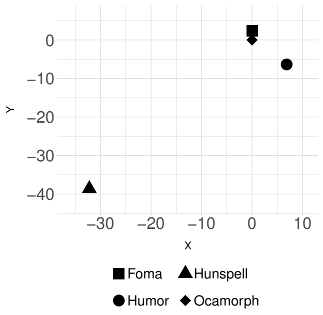

Figure 4: Visualizing recognition distances

7.3. Equivalent morphological structures

The third important question is how many words are there whose morphological structure is determined in the same way by more than one morphological analyzers. In section 6 we read about the different mappings among the token systems of the four tools, so it is interesting to check if we convert every token to a common token system, are the results the same?

Table 9 shows the number of equivalent words, common words and the percentage ratio. We chose the token system of Hunmorph-Ocamorph, and executed the mapping using the tables in the appendix[11].

Morphological Analyzers	**Equivalent Structures**	**Recognized Words**	**Percentage Ratio**
Ocamorph-Foma	1 818 027	2 342 940	77.60%
Ocamorph-Humor	334 617	357 182	93.68%
Ocamorph-Hunspell	53 320	57 659	92.47%
Foma-Humor	363 735	416 310	87.37%
Foma-Hunspell	50 569	74 728	67.67%
Humor-Hunspell	16 755	17 975	93.21%
Ocamorph-Foma-Humor	279 327	344 951	80.98%
Ocamorph-Foma-Hunspell	37 949	56 812	66.80%
Ocamorph-Humor-Hunspell	7 685	8 711	88.22%
Foma-Humor-Hunspell	8 240	10 266	80.26%
Ocamorph-Foma-Humor-Hunspell	6 745	8 612	78.32%

Table 9: The number of equivalently analyzed words, common words and the percentage ratio across the different analyzer combinations

The overall results are the same as before: Ocamorph and Foma are close to each other as 77.60% of the commonly recognized words have the same morphological structure as well. Although the percentage ratios are higher for other combinations, the number of common words are much lower, so in reality the number of equivalent morphological structures is still much lower for Humor and Hunspell.

We also analyzed the differences manually by selecting 100 problematic words randomly from every set and checking what causes the differences. First, we realized again that with Foma, sometimes the stem is invalid, but also we found cases where the POS category was recognized in a different, invalid way. Sometimes nouns were marked as adjectives, and so on. Another big difference was that the concrete derivation tokens missing from Foma, Humor and Hunspell were sometimes replaced with the target POS category. The final typical error was that with some verb features, Foma, Humor and Hunspell sometimes added an extra $\langle PERS \rangle$ token at the end of the verb structures in case of normal 3rd person verbs, while Ocamorph didn't do that.

Moreover, Hunspell's structures contained many unnecessary pieces of information that caused our parser to have a hard time determining the correct structure of the word in case of compound words with many affix types.

In case of derivation tokens missing from analyzers other than Ocamorph, the results were incorrect in most cases, as the weaker analyzers determined the stem to be the derived word form.

The similarity and distance values defined in equations 3.5 and 3.6 can be seen in Table 10.

[11]`http://users.iit.uni-miskolc.hu/~szabo84/articles/morphological-analyzers/appendix.pdf`

A_1	A_2	$\sigma^E_{1,2,\chi_{1,2}}$	$\delta^E_{1,2,\chi_{1,2}}$
Ocamorph	Foma	0.7759	1.2887
Ocamorph	Humor	0.9188	1.0883
Ocamorph	Hunspell	0.7969	1.2549
Foma	Humor	0.8727	1.1458
Foma	Hunspell	0.4497	2.2237
Humor	Hunspell	0.7372	1.3564

Table 10: Similarity ($\sigma^E_{i,j,\chi_{i,j}}$) and distance ($\delta^E_{i,j,\chi_{i,j}}$) values among the morphological analyzers based on their equivalence

Figure 5 shows the distance of the four morphological analyzers based on the morphological structure equivalence of their recognized words. Ocamorph and Foma are close together, Humor and Hunspell are a bit further, as we can see in the figure.

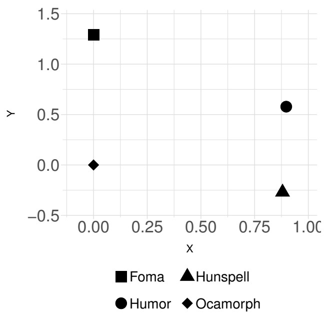

Figure 5: Visualizing equivalence distances

7.4. Cumulative Distance

To visualize the overall cumulative distance among the four examined morphological analyzers, we defined the formula in equation 3.7. Figure 6 shows the overall distances based on this final metric.

All in all we can see that Hunspell is far away from the remaining three morphological analyzers. Hunmorph-Ocamorph and Hunmorph-Foma are the closest, while Humor is close to the two Hunmorph based systems, preceding Hunspell by far.

Based on the previous statistics, we can state that the best morphological analyzer is Hunmorph-Ocamorph. Although Hunmorph-Foma can also be used successfully, the Ocamorph based tool is better regarding the total number of recognized words and the accuracy of the produced morphological structure. Behind Hunmorph-Foma the number of recognized words is the second best, but as we

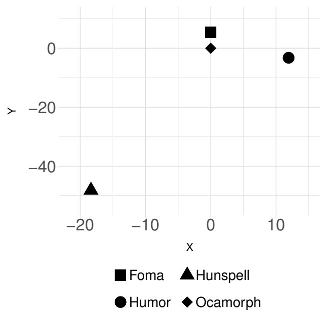

Figure 6: Visualizing cumulative distances

found multiple cases where Hunmorph-Foma returned invalid stems, Hunmorph-Ocamorph seems more reliable.

8. Conclusion

In this paper we benchmarked the four most widely used morphological analyzer tools for the Hungarian language: Hunmorph-Ocamorph, Hunmorph-Foma, Humor and Hunspell. For evaluation, we had these tools analyze more than 13 million Hungarian word candidates. As the four annotation token systems were different, we created a complete mapping among them, first by using a novel automated algorithm, then also manually, to evaluate the automated results. After mapping the tokens, we compared the morphological analyzers based on different statistics: the main differences in the annotation token systems, the total number of words that got recognized by each analyzer and the number of recognized words, finally we looked at how many words had equivalent morphological structure across the four analyzers. For each of these statistics, we defined the concept of similarity and distance, and besides calculating these metrics, we also visualized the distances among the morphological analyzers. From the three statistics and also a fourth, cumulative distance metric it became visible that Hunmorph-Ocamorph and Hunmorph-Foma recognized many words and had many things in common, while Humor and Hunspell had worse results. From the two Hunmorph based tools it seems that Hunmorph-Ocamorph is worth using as Hunmorph-Foma sometimes provided invalid lemmas. In the future we will use the results of Hunmorph-Ocamorph to generate Hungarian word pairs for each Hungarian affix type. This data set will be used as training data for our novel inflection/lemmatization method.

Acknowledgements. The described article/presentation/study was carried out as part of the EFOP-3.6.1-16-00011 "Younger and Renewing University – Innovative Knowledge City – institutional development of the University of Miskolc aiming at intelligent specialisation" project implemented in the framework of the Szechenyi 2020 program. The realization of this project is supported by the European Union, co-financed by the European Social Fund.

References

[1] Bauer, L., Introducing linguistic morphology, *Edinburgh University Press Edinburgh* (2003).
`https://doi.org/10.1017/s0022226700014638`

[2] Endrédy, I., Corpus based evaluation of stemmers, *Uniwersytet im. Adama Mickiewicza w Poznaniu* (2015).

[3] Endrédy, I., Novák, A., Szótövesítők összehasonlítása és alkalmazásai, *Alkalmazott Nyelvtudomány*, Vol. 15 (2015), 7–27.

[4] Gelbukh, A., Alexandrov, M., Han, S-Y., Detecting inflection patterns in natural language by minimization of morphological model, *Iberoamerican Congress on Pattern Recognition* (2004), 432–438.
`https://doi.org/10.1007/978-3-540-30463-0_54`

[5] Hulden, M., Foma: a finite-state compiler and library, *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics: Demonstrations Session* (2009), 29–32.
`https://doi.org/10.3115/1609049.1609057`

[6] McCandless, M., Hatcher, E., Gospodnetic, O., Lucene in action: covers apache lucene 3.0, *Manning Publications Co.* (2010).

[7] Minnen, G., Carroll, J., Pearce, D., Robust, applied morphological generation, *Proceedings of the first international conference on Natural language generation*, Vol. 14 (2000), 201–208.
`https://doi.org/10.3115/1118253.1118281`

[8] Minnen, G., Carroll, J., Pearce, D., Applied morphological processing of English, *Natural Language Engineering*, Vol. 7 (2001), 207–223.
`https://doi.org/10.1017/s1351324901002728`

[9] Pirinen, T., Lindén, K. and others, Creating and weighting hunspell dictionaries as finite-state automata, *Investigationes Linguisticae*, Vol. 21 (2010), 1–16.
`https://doi.org/10.14746/il.2010.21.1`

[10] Porter, M. F., An algorithm for suffix stripping, *Program*, Vol. 14 (1980), 130–137.
`https://doi.org/10.1108/eb046814`

[11] Porter, M. F., Boulton, R., Snowball stemmer, Online `http://www.snowball.tartarus.org`, (2001).

[12] Prószéky, G., Kis, B., A unification-based approach to morpho-syntactic parsing of agglutinative and other (highly) inflectional languages, *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics* (1999), 261–268.
`https://doi.org/10.3115/1034678.1034723`

[13] Prószéky, G., Tihanyi, L., Humor: High-speed unification morphology and its applications for agglutinative languages, *La tribune des industries de la langue* (1993), 28–29.

[14] Tordai, A., De Rijke, M., Four stemmers and a funeral: Stemming in Hungarian at clef 2005, *Workshop of the Cross-Language Evaluation Forum for European Languages* (2005), 179–186.
https://doi.org/10.1007/11878773_20

[15] Trón, V., Halácsy, P., Rebrus, P., Rung, A., Vajda, P., Simon, E., Morphdb. hu: Hungarian lexical database and morphological grammar, *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06)* (2006).

[16] Trón, V., Kornai, A., Gyepesi, Gy., Németh, L., Halácsy, P., Varga, D., Hunmorph: open source word analysis, *Proceedings of the Workshop on Software* (2005), 77–85.
https://doi.org/10.3115/1626315.1626321