Computing and Informatics, Vol. 38, 2019, 963–985, doi: 10.31577/cai_2019_4_963

OPTIMIZATION OF THE MORPHER MORPHOLOGY ENGINE USING KNOWLEDGE BASE REDUCTION TECHNIQUES

Gábor Szabó, László Kovács

Institute of Information Technology
University of Miskolc
Miskolc-Egyetemváros, H 3515, Hungary
e-mail: szgabsz91@gmail.com, kovacs@iit.uni-miskolc.hu

Abstract. Morpher is a novel morphological rule induction engine designed and developed for agglutinative languages. The Morpher engine models inflection using general string-based transformation rules and it can learn multiple arbitrary affix types, too. In order to scale the engine to training sets containing millions of examples, we need an efficient management of the generated rule base. In this paper we investigate and present several optimization techniques using rule elimination based on context length, support and cardinality parameters. The performed evaluation tests show that using the proposed optimization techniques, we can reduce the average inflection time to 0.52 %, the average lemmatization time to 2.59 % and the number of rules to 2.25 % of the original values, while retaining a high correctness ratio of 98 %. The optimized model can execute inflection and lemmatization in acceptable time after training millions of items, unlike other existing methods like Morfessor, MORSEL or MorphoChain.

Keywords: Machine learning, natural language processing, inflection, lemmatization, agglutination, morphology, optimization, rule base reduction

Mathematics Subject Classification 2010: 68T50

1 INTRODUCTION

Natural language processing is one of the actively researched scientific areas nowadays. One of the goals of these research projects is to create automated methods

for processing free texts. Developing efficient NLP applications requires processing modules on several abstraction layers, among which the lowest layer is morphology.

Morphology deals with the inner components of words called morphemes, that are the smallest units of the language with meaning [1]. The grammatically correct root form of a word is called a lemma, whose base meaning can be modified by adding affixes to it. These affixes can appear before the lemma (prefix), after the lemma (suffix) or even inside the word (infix). The complexity of automated morphological processing of words comes from the fact that in some cases adding affixes can modify the root form as well, e.g. *(try, tried)*. Inflection is the operation that produces the inflected form from the lemma, while lemmatization determines the lemma and the list of affix types in an arbitrary word.

Historically the first successfully applied, popular model for inducing inflection rules of agglutinative languages was the two-level morphology model [6]. The name of the model comes from the fact that the inflected word forms are represented on two levels: the surface level stores the written form, and the lexical level stores the morphological structure. Dictionaries were used to collect the valid lemmas and affix types of the target language, while FSTs (finite-state transducers) were trained to apply the required transformations on the input words. The FST model appears in several other publications as well, because this model fits the need of transforming one word to another based on some pre-learnt rules. One of the most widely used FST category is the subsequential transducer model [9, 10] that can be trained using the OSTIA algorithm [4, 12].

A very simple transformation engine called the tree of aligned suffix rules (TASR) is proposed in [15]. It is a supervised model that generates elementary rules from the training word pairs and stores them in a tree. Unfortunately the TASR model can only handle suffix rules and not prefix or infix rules. According to the experiments of [7], for languages that contain only suffix rules, the TASR model can be used very efficiently.

A more recent unsupervised segmentation model is Morfessor [3, 20]. This model uses a statistical training method to determine the morpheme boundaries inside the words. One downside of the original Morfessor model is that it only uses global frequencies and not local probabilities, so it does not take into account which morpheme can come after which other morphemes. However, there are several other methods that either use the Morfessor model or extend it.

The MorphoChain engine [11] is one such extension of Morfessor. The main addition of this improved model is that it also uses orthographic and semantic views of the input words, thus improving the segmentation correctness. Another improvement was presented in [2] that adapts the MorphoChain segmentation system in a way that it can identify identical morphemes with spelling differences. Based on the results, this model outperforms both the original MorphoChain system and MORSEL [8].

The target language of our research is Hungarian, that is highly agglutinative and morphologically complex language. It has many affix types and each word can contain multiple affixes. Most of the affix types are suffixes, but there are a couple

of prefix and prefix-suffix affix types as well, like comparative and superlative[1]. In many cases, adding an affix to a word also changes some characters in the base form.

As an example, *meg|szent|ség|telen|ít|hetetlen|ség|es|ked|és|eitek|ért* is one of the longest Hungarian artificial words that means *"for your [plural] continued behaviour as if you could not be desecrated"*. It contains 11 affixes. On the other hand, one lemma might have several inflected forms: in our data set, the word *konfigurál* (*configure*) has 86 different forms in our data set, including *átkonfigurálom* (*I reconfigure it*), *konfigurálásukkal* (*with their configuration*), etc. Such examples show the complexity of the target language.

For Hungarian, Hunmorph-Ocamorph [18], Hunmorph-Foma [5], Humor [14] and Hunspell [13] are four popular morphology tools. These analyzers can determine all the possible morphological structures of input words, including their lemma and the list of affix types found in the input word. According to the analysis of these tools [16], Hunmorph-Ocamorph is the most advanced among them. Its engine (Ocamorph) is language independent, and Morphdb.hu [19] is the language dependent knowledge database that stores the set of affix types and their related transformation rules. Hunmorph-Ocamorph can recognize more than 4 million words using the lexical database of Morphdb.hu, but unfortunately this database has been constructed by human experts, and not learnt by an automated learning algorithm.

The Morpher[2] system [17] is a morphology model that can statistically learn prefix, suffix and infix transformation rules from a training set containing (word, lemma, part of speech, morphosyntactic tags) tuples. The model is suitable for complex agglutinative languages like Hungarian, and it can handle multiple affix types. According to the evaluation, Morpher can learn 100 000 training items in about 4 seconds, then execute inflection and lemmatization in about 2.4 milliseconds and 2.4 seconds, respectively, reaching about 97 % of average correctness ratio for previously unseen words. The known limitation of the model is that the lemmatization time increases exponentially as we increase the training data set.

The main goal of this paper is to perform space and time complexity analysis of the baseline Morpher model and introduce several optimization techniques for rule base reduction so that the engine can scale more easily to large training data sets containing millions of items.

2 THE MORPHER MODEL

The training data of the Morpher model is a D_{train} set containing training items in the form of $\left(w, \bar{w}, \bar{T}_0, \langle T_i \rangle_{i=1}^{k}\right)$, where $w \in W$ is the inflected form, $\bar{w} \in \bar{W}$ is the lemma of w, $\bar{T}_0 \in \bar{\mathbb{T}}$ is the part of speech, and $\langle T_i \rangle_{i=1}^{k}$ is the ordered list of k affix types in w, $T_i \in \mathbb{T}$.

[1] *Jó, jobb, legjobb* means *good, better, best.*

[2] `https://github.com/szgabsz91/morpher`

From this training data, Morpher can learn string-based inflection and lemmatization rules, the conditional probabilities of the affix types, as well as the valid lemmas and their parts of speech.

Figure 1 displays the overall structure of the model.

Manager: manages the inflection and lemmatization tasks by coordinating the transformation engines generated for each affix type. Has a knowledge about the valid lemmas of the target language and the conditional probabilities of the affix type chains.

Transformation engines: one transformation engine can learn the transformation rules of a single affix type. For every affix type, a separate transformation engine is generated and trained.

Probabilities: during the training phase, all the possible affix type chains are analyzed and the conditional probabilities are stored and updated so that the manager knows which affix type can come after which other affix types and with how much probability.

Lemmas: all the valid lemmas are stored, as well as their associated parts of speech so that during lemmatization the engine knows when it can stop processing an affix type chain candidate.

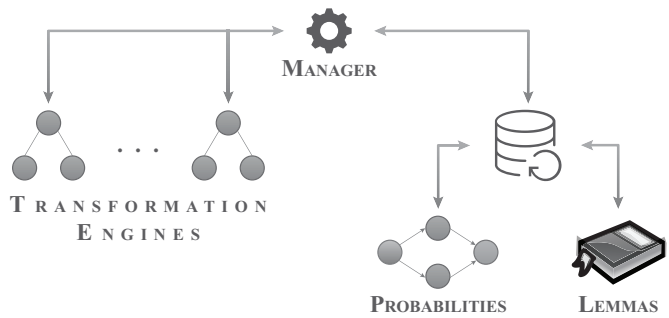


Figure 1. The main components of the Morpher model

2.1 Transformation Engine

The responsibility of the transformation engine model is to learn simple string-based transformation rules from word pairs of an affix type induced by the manager. The transformation engines can inflect and lemmatize input words based on the induced transformation rules. We build a transformation engine for every affix type of the target language.

During the training phase, after the word pairs for an affix type are induced by the manager, we identify the changing parts of the base forms, and generate

a number of atomic rewrite rules for them in the form of $R = (\alpha, \sigma, \tau, \omega)$, where α is the prefix, σ is the changing part of the base form, τ is the replacement string and ω is the suffix. The first atomic rule is called the core atomic rule $\hat{R}$ for which $|\alpha| = |\omega| = 0$. The other rules are extended from this core atomic rule by prepending and appending one character at a time, from the words to the context, essentially filling in α and ω.

For each atomic rule, we calculate different statistics, including the support value and the word frequency. The support of a rule equals the number of training word pairs that the rule matches, while the word frequency is the sum of frequencies of the related training words. A word's frequency is equal to its number of occurrences in the input free text sources, from which the training data was generated.

During inflection and lemmatization, the task is to find the best matching atomic rules for the input word. For this, we define a fitness function that returns the goodness value of an atomic rule R for the input word w. This fitness function is

$$f(R \mid w) = \frac{|\gamma(R)|}{|w|} \cdot \delta(\gamma(R), w)$$

where

- R is the atomic rewrite rule in question,
- w is the input word that needs to be inflected or lemmatized,
- $\gamma(R)$ is the context of the atomic rewrite rule,
- δ is a function that either returns 0 if the context is not found in the input word, or 1 otherwise. (In this sense, it is similar to the Kronecker delta, but could be implemented differently.)

The rule context is $\alpha+\sigma+\omega$ during inflection and $\alpha+\tau+\omega$ during lemmatization. Using the fitness function, we can select the matching atomic rewrite rules for the input word.

2.2 Conditional Probabilities

During the training phase, the Morpher model learns all the possible affix type chains, and their conditional probabilities. M is the function that can return the probability of an affix type chain:

$$M\left(\bar{T}_0, T_1, \ldots, T_i\right) = \begin{cases} P\left(\bar{T}_0\right), & \text{if } i = 0, \\ P\left(\bar{T}_0\right) \cdot \prod_{j=1}^{i} P\left(T_j \mid \bar{T}_0, T_1, \ldots, T_{j-1}\right), & \text{if } i = 1, 2, \ldots \end{cases}$$

We use the relative frequencies in the training data set D_{train} for probability calculation.

2.3 Manager

The manager submodule coordinates all the other submodules to learn the required morphological features of the training word pair set, as well as perform inflection and lemmatization, producing complex responses with multiple steps.

The input of the inflection operation is a lemma $\bar{w}_0$ and a set of affix types $\{T_1, T_2, \ldots, T_k\}$. The output is a set of n items in the form of

$$\left(\bar{T}_0, \langle (T_i, w_i) \rangle_{i=1}^{m}, \vartheta\right)$$

containing

- the $\bar{T}_0$ part of speech,
- the (T_i, w_i) steps, where $\langle T_i \rangle_{i=1}^{m}$ is a valid permutation of the input affix types according to M, and $\langle w_i \rangle_{i=1}^{m}$ are the produced inflected forms, and
- the ϑ aggregated weight of the response.

The responses are sorted by ϑ, in a descending order.

Similarly, the input of the lemmatization operation is an arbitrary inflected word form w, and the output is a set of n items in the form of

$$\left(\langle (T_i, w_i) \rangle_{i=m}^{1}, \bar{T}_0, \vartheta\right)$$

containing

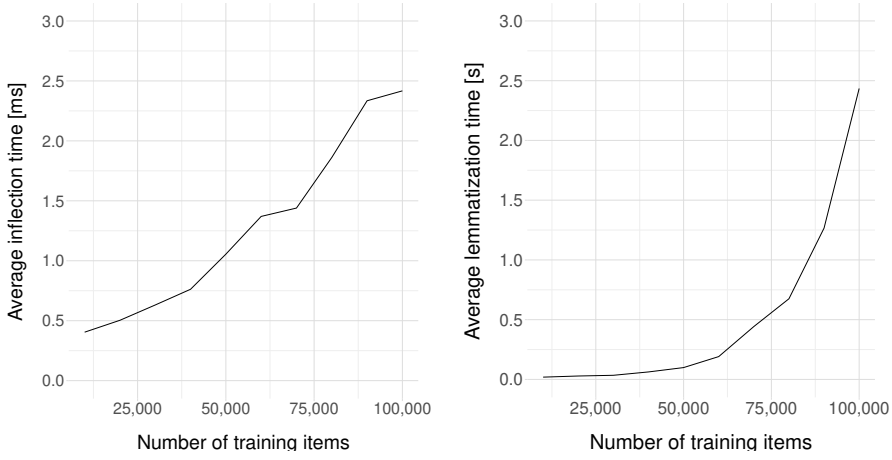
- the (T_i, w_i) steps ($w_m = w$), where $\langle T_i \rangle_{i=m}^{1}$ is a valid affix type chain, and $\langle w_i \rangle_{i=m}^{1}$ are the produced base forms,
- the $\bar{T}_0$ part of speech, and
- the ϑ aggregated weight of the response.

The responses are sorted by ϑ, in a descending order. The ϑ aggregated weight is calculated using the normalized affix type conditional probability, and the aggregated fitness of the output words in the steps.

3 OPTIMIZATION TECHNIQUES

Morpher's generalization ability seemed to be promising, reaching about 97 % of average correctness ratio in case of a training data set containing 100 000 random items and evaluating 10 000 previously unseen random words. On the other hand, we can see a linear increase of average inflection time and an exponential increase of lemmatization time, as shown in Figures 2 a) and 2 b).

This increase is due to the nature of the problems. For example the exponential increase of the lemmatization process is caused by the fact that the manager needs to try all the possible preceding affix type candidates at every affix type. This cannot be changed, because we do not know the exact set of affix types found in the



a) Average inflection time b) Average lemmatization time

Figure 2. Average inflection and lemmatization times of the baseline Morpher model

input word. However, we can extend the number of training items with which the Morpher system can perform inflection and lemmatization in acceptable time.

In order to reduce the size of the knowledge base, we need to modify the training phase of the individual transformation engines associated with the affix types of the target language, since the other components of the Morpher model (the lemma database and the conditional probability store) cannot lose information without a significant loss of correctness ratio.

3.1 Eliminating the Redundant Atomic Rewrite Rules

The main idea behind this optimization technique is to drop the redundant atomic rules that are covered by other rules in the rule base.

Definition 1 (Redundant atomic rule)**.** The atomic rule $R_i = (\alpha_i, \sigma_i, \tau_i, \omega_i)$ is a redundant rule if and only if there exists another $R_j = (\alpha_j, \sigma_j, \tau_j, \omega_j)$ atomic rule in the rule base such that $\gamma(R_j) \subseteq \gamma(R_i)$, $\sigma_i = \sigma_j$ and $\tau_i = \tau_j$. In this case we say R_i is covered by R_j.

As an example, the contexts of $R_1 = (\text{alm}, \text{a}, \text{át}, \#)$[3] and $R_2 = (\text{-}, \text{a}, \text{át}, \#)$ are $\gamma(R_1) = \text{alma}\#$ and $\gamma(R_2) = \text{a}\#$, respectively. If both rules are in the rule base, we can say that R_1 is redundant, since $\gamma(R_2) \subseteq \gamma(R_1)$ and the transformation (σ and τ components) are also the same, i.e. R_1 is covered by R_2.

On the other hand, if $R_3 = (\text{toll}, \text{-}, \text{at}, \#)$ and $R_4 = (\text{l}, \text{-}, \text{t}, \#)$ are part of the same rule base, they do not cover each other. Although $\text{l}\# = \gamma(R_4) \subseteq \gamma(R_3) = \text{tollat}\#$,

[3] # is the special word-end symbol.

R_3 is not redundant, because the transformations are different: R_3 appends *'at'* at the end of the word, while R_4 only appends *'t'*.

The learning algorithm can detect the redundant rules during the rule generation process, and it immediately blocks their generation. This means that the redundant rules are not stored in the rule base, and we do not have to execute a second wave of rule filtering after the original training phase. We introduce a new p_{max} optimization parameter. If for an arbitrary word the atomic rules $\hat{R}_1$, R_2, ..., R_k were generated ($\hat{R}_1$ being the core atomic rule), then using p_{max} only $\hat{R}_1$, R_2, ..., R_l will be retained, where $l = min\,(p_{max}, k)$. Those atomic rewrite rules that have a longer context than R_l are simply omitted.

Proposition 1. Using $p_{max} = 1$, storing only $\hat{R}_1$ for each word pair and dropping the other $R_2, \ldots, R_k$ extended atomic rules is equivalent with generating every possible atomic rule and then dropping all the redundant atomic rules.

Proof. According to Definition 1, an atomic rule $R_i = (\alpha_i, \sigma_i, \tau_i, \omega_i)$ is redundant if and only if there exists another atomic rule $R_j = (\alpha_j, \sigma_j, \tau_j, \omega_j)$ such that $\gamma\,(R_j) \subseteq \gamma\,(R_i)$, $\sigma_i = \sigma_j$ and $\tau_i = \tau_j$.

We can assume indirectly that by executing the first part of the proposition, there remains at least one redundant atomic rule $\tilde{R} = (\tilde{\alpha}, \tilde{\sigma}, \tilde{\tau}, \tilde{\omega})$. This means that there is at least one other atomic rule $R = (\alpha, \sigma, \tau, \omega)$ such that $\gamma\left(\tilde{R}\right) \subseteq \gamma\,(R)\,, \tilde{\sigma} = \sigma$ and $\tilde{\tau} = \tau$.

From these formulae, we can see that $\gamma\left(\tilde{R}\right) = \tilde{\alpha} + \tilde{\sigma} + \tilde{\omega} \subseteq \alpha + \sigma + \omega = \gamma\,(R)$ and since $\tilde{\sigma} = \sigma$, we can see that $\tilde{\alpha} + \sigma + \tilde{\omega} \subseteq \alpha + \sigma + \omega$.

This means that $|\tilde{\alpha} + \sigma + \tilde{\omega}| \leq |\alpha + \sigma + \omega|$. There are two cases to check:

- If $|\tilde{\alpha} + \sigma + \tilde{\omega}| = |\alpha + \sigma + \omega|$, then $\tilde{R} = R$ (as all the components are equal due to both rules being core atomic rules due to $p_{max} = 1$), so $\tilde{R}$ is a non-redundant item in the rule database.
- If $|\tilde{\alpha} + \sigma + \tilde{\omega}| < |\alpha + \sigma + \omega|$, then $|\tilde{\alpha}| = |\tilde{\omega}| = |\alpha| = |\omega| = 0$ since both $\tilde{R}$ and R are core atomic rules. This means that $|\sigma| < |\sigma|$, which is a contradiction.

From both cases we get a contradiction, which means that the proposition is true. □

3.2 Limiting the Generalization Factor

The potential problem with p_{max} optimization, especially using $p_{max} = 1$ is that we only retain atomic rewrite rules with very short contexts. This means that the matching rules for the input word might have very different $\sigma \Rightarrow \tau$ transformations, increasing the number of outputs at each affix type step, making the inflection and lemmatization processes extremely slow. This effect is called overgeneralization and can be prevented by also retaining some redundant rules to increase the information in the system.

In order to avoid this overgeneralization effect, we introduce another parameter called p_{gen}, that identifies the minimum context length of the generated atomic rewrite rules. This means that for all the retained atomic rules, $\gamma(R) \geq p_{gen}$. Every other atomic rule is dropped during the rule generation process.

This way we can limit the number of atomic rules from below, while p_{max} is an upper limit on the atomic rule context length. We can also combine these two parameters, retaining a slice of all the possible atomic rewrite rules. For example, $p_{max} = 2$ and $p_{gen} = 3$ will retain rules whose context contains at least two characters, but only 3 of these rules per each word pair. In this way, we drop the most general rules ($\gamma(R) = 1$ and $\gamma(R) = 2$), and also drop the most specific rules where $\gamma(R) \geq 2 + 3 = 5$ in case of suffix rules.

3.3 Indirect Data Cleaning

While the p_{max} and p_{gen} parameters drop the atomic rewrite rules based on their contexts, we can also drop some rules based on the statistical attributes of the training data set.

For each atomic rule, we calculate a support value and a word frequency value, as described in Section 2. The support value is the number of training word pairs that the rule matches, while the word frequency is the sum of frequencies of the related training words. A word's frequency is equal to the number of occurrences in the input free text sources, from which the training data was generated.

For the support and word frequency based elimination method, we introduce the p_{supp} and p_{freq} parameters that drop every atomic rule whose support is less than p_{supp}, or whose word frequency is less than p_{freq}.

This optimization method is based on the widely used frequency based reduction concept that can be found in other research areas as well, such as association rule mining, where only frequent item sets are considered during rule generation. This also means that we perform an indirect data cleaning, since rare rules apply to fewer words. As the training data is generated automatically, it can contain words with typos or otherwise meaningless words that can be omitted using these two parameters.

4 SPACE AND TIME COMPLEXITY ANALYSIS

4.1 Space Complexity

The number of transformation engines is equal to the number of affix types, so the space complexity is $\Omega(|\mathbb{T}|)$.

The number of conditional probability values for inflection is equal to the number of valid affix type orders: $\Omega\left(\left|\left\{\left(\bar{T}_{i_0}, T_{i_1}, \ldots, T_{i_k}\right) \mid M\left(\bar{T}_{i_0}, T_{i_1}, \ldots, T_{i_k}\right) > 0\right\}\right|\right)$.

The upper size limit of the lemma database can be approximated with the number of training items. In the worst case, every word in the training data set has different lemmas, and as such, the size complexity is $O(|D_{\text{train}}|)$.

The number of generated rules depends on the number of word pairs for the related affix type from D_{train}. In the worst case, it equals the number of training items where the last affix type is the associated affix type of the transformation engine. Therefore the number of word pairs generated to train the transformation engine of the affix type T can be approximated with $O\Big(\Big|\Big\{\Big(w, \bar{w}, \bar{T}_0, \langle T_i\rangle_{i=1}^{k}\Big) \in D_{\text{train}} \mid T_k = T\Big\}\Big|\Big)$.

For every deduced word pair we generate a number of atomic rules. The approximation of the number of generated atomic rewrite rules for the word pair (w_1, w_2) is $\max(|w_1|, |w_2|) - |\sigma|$ without any optimizations. For the whole transformation engine related to the affix type T, the approximation of the generated atomic rewrite rules is

$$O\left(\left|\left\{\left(w, \bar{w}, \bar{T}_0, \langle T_i\rangle_{i=1}^{k}\right) \in D_{\text{train}} \mid T_k = T\right\}\right| \cdot \left(\max\left(|w_{j_1}|, |w_{j_2}|\right) - |\sigma_j|\right)\right) \tag{1}$$

where the j index refers to the word pair for which the right component is maximal.

The optimization techniques introduced in Section 3 optimize the right component of the above formula. The p_{max} optimization parameter (Subsection 3.1) makes sure that the right component is at most p_{max}:

$$O\left(\left|\left\{\left(w, \bar{w}, \bar{T}_0, \langle T_i\rangle_{i=1}^{k}\right) \in D_{\text{train}} \mid T_k = T\right\}\right| \cdot \min\left(p_{max}, \max\left(|w_{j_1}|, |w_{j_2}|\right) - |\sigma_j|\right)\right). \tag{2}$$

Using $p_{max} = 1$, this formula will be as simple as

$$O\left(\left|\left\{\left(w, \bar{w}, \bar{T}_0, \langle T_i\rangle_{i=1}^{k}\right) \in D_{\text{train}} \mid T_k = T\right\}\right|\right).$$

The p_{gen} optimization parameter (Subsection 3.2) will result in a space complexity of

$$O\left(\left|\left\{\left(w, \bar{w}, \bar{T}_0, \langle T_i\rangle_{i=1}^{k}\right) \in D_{\text{train}} \mid T_k = T\right\}\right| \cdot \left(\max\left(|w_{j_1}|, |w_{j_2}|\right) - |\sigma_j| - \Upsilon_{gen}\right)\right) \tag{3}$$

where Υ_{gen} denotes the minimum number of generated atomic rules that have a context shorter than p_{gen} among the training word pairs.

Asymptotically this means that in the worst case, no reduction occurs, if all the generated atomic rules have at least as long context as p_{gen}. Additionally the redundant rules are eliminated after processing each word pair or after each training iteration. However, this cannot be estimated, since the final number of retained atomic rules depends on the quality of D_{train}.

The space complexity of the p_{supp} and p_{freq} optimization parameters, i.e. how many atomic rewrite rules are retained by them, depends greatly on the training data set D_{train}. While the support value only refers the number of words for which the given atomic rule is generated, the word frequency also contains information about the free text sources from which the training items in D_{train} were constructed.

4.2 Time Complexity

The training phase consists of three main parts:

- Steps with $O(1)$ time complexity like storing the lemmas or updating the conditional probability values for each training item.
- Generating word pairs from the training items for each affix type.
- Generating atomic rewrite rules from the training word pairs for each affix type.

The generation of word pairs can be done in $O(|D_{\text{train}}|^2)$ time, since we need to find all the possible item pairs that are adjacent in the affix type chains. However, we can improve the pairing algorithm by only going through the items with the same lemma as the item under processing. This way the majority of the search space remains untouched. Also, for the set of items that have only one affix type, we can generate a word pair using the lemma and the word of the same item without any searching, in $O(1)$ time.

For every training word pair, we first have to find the core. This can be done in $O(\max(|w_1|, |w_2|))$ time for the word pair (w_1, w_2). Then we need to generate the required atomic rewrite rules. Every rule can be generated in $O(1)$ time, so the approximation depends on the number of generated atomic rewrite rules, see Formulae (1), (2) and (3). Without any optimization, the whole generation process can be done in $O(\max(|w_{j_1}|, |w_{j_2}|) - |\sigma_j|)$ time per word pair. This can be reduced to $O(p_{max})$ using p_{max} optimization that will result in $O(1)$ time complexity using $p_{max} = 1$. Using p_{gen} optimization, the generation time of the atomic rules will also be $O(\max(|w_{j_1}|, |w_{j_2}|) - |\sigma_j| - \Upsilon_{gen})$. The time complexity of the p_{supp} and p_{freq} optimization parameters cannot be approximated accurately, since the number of retained atomic rules depends on the quality of the training data.

The first task during inflection is to generate all the valid orders of the given k affix types. This can be done in at most $O(k!)$ steps. In the worst case, all the possible permutations are valid, and for each permutation we have to check if the chain's conditional probability is positive, which can be done in $O(1)$. For every possible valid order, we need to go through the affix type chain and perform local inflection based on the atomic rewrite rules of the appropriate affix types one by one. We assume that applying an atomic rule on a word and checking if an atomic rule matches a word can be done in constant time, so at every affix type the generation of the inflected forms can be approximated with $O(|\{R\}|)$, which will be either Formula (1), (2) or (3) as we saw earlier, based on the applied optimization techniques.

As for lemmatization, the provided input does not contain any information about how many and which affix types will appear in the word to lemmatize. In the worst case, there may be $O(|\mathbb{T}|)$. At every step, the number of atomic rules to process and potentially apply can be approximated with Equation (1), (2) or (3) based on the optimization technique we used during training. The number of previous affix types

that need to be checked is $O(|\mathbb{T}| - 1)$. This means that all in all lemmatization can be done in exponential time, roughly in $O(|D_{\text{train}}|^{|D_{\text{train}}|})$ time at most.

Although the size of the knowledge base might seem to be the biggest reason of the increase of average inflection and lemmatization costs, there are also other factors to be considered. For example if we eliminate too many redundant atomic rules with longer contexts, only atomic rules with shorter contexts will remain in the rule base. This can cause many rules to match the input words, resulting in many inflection and lemmatization responses for each affix type. Having long affix type chains, the execution time can increase combinatorically.

5 TEST SYSTEM

For the evaluation of the baseline Morpher model and the optimization techniques, we implemented a test system using the Java 11 programming language. With the modern language features we could introduce clean interfaces among the different submodules, using the modularization techniques introduced by Java 9. Moreover parallel streams were used for processing large amounts of data in parallel, in a functional manner. These features make the implemented system more maintainable and efficient.

Figure 3 presents the data pipeline of the evaluation test system.

- Initially, a large number of Hungarian words were collected from the web using the site of the National Széchenyi Library[4].
- Hunmorph-Ocamorph [18] was used to analyze these words, creating a pre-annotated corpus.
- The corpus was fed to the data generator that emitted both training data and evaluation data.
- The training data was given to the trainer submodule that returned a trained instance of the model.
- The evaluator submodule received the trained model and the evaluation data, and performed several tests to evaluate the model against different metrics.

The number of sample word pairs generated by this data pipeline was 3 612 494.

6 EVALUATION

We examine several metrics during evaluation so that we can compare the baseline Morpher method with other existing models, and evaluate the optimization parameters. These metrics include

- the average training, inflection and lemmatization time,

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

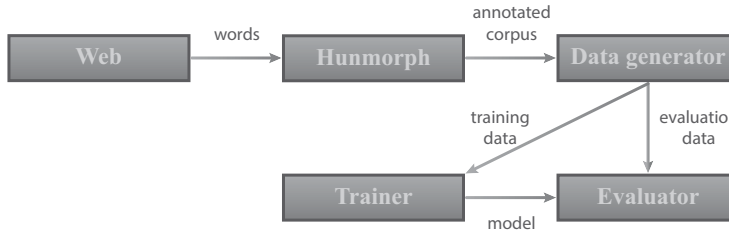


Figure 3. Test system pipeline

- the average number of responses and the average index of the expected response,
- the average correctness ratio,
- the average number of atomic rewrite rules and the average size of the knowledge base.

All of these metrics should be minimized except for the average correctness ratio. The performed tests are performed using the following steps:

1. Generate n_1 training items and train the model.
2. Generate n_2 evaluation items and evaluate the model using the previously mentioned metrics.
3. Repeat the test n_3 times and calculate the average of the examined metrics.

The n_1 parameter is increased from 10 000 to 100 000 with 10 000 increments in Subsections 6.1 and 6.2, while it is increased from 500 000 to 3 million with 500 000 increments in Subsection 6.3. The n_2 parameter is 10 000 in all cases, and n_3 is chosen to be 10. Every training and evaluation item is chosen randomly for each test scenario, but the training and evaluation item sets are always disjoint.

The test machine is a Macbook Pro with 3.1 GHz Intel Core i7 processor and 16 GB memory.

6.1 Comparing the Baseline Morpher Model with Other Morphology Engines

To compare the baseline Morpher model with existing methods, we executed the same evaluation tests on the baseline Morpher model, Morfessor, MORSEL, MorphoChain and Hunmorph-Ocamorph. Since the interface and functionality of these tools differ in some points, we could not perform all the tests on every existing methods:

- MorphoChain failed with an *OutOfMemoryError* using 50 000 training items, so we dropped this tool, since we could not compare its final metric values.
- Morfessor and MORSEL are segmentation tools, so they could not perform inflection.

- Since we used Hunmorph-Ocamorph for training data preprocessing, it did not make sense to include it in the comparisons other than the database size.

Figure 4 displays the average correctness ratio of the investigated methods. The baseline Morpher method's correctness ratio increases from about 73 % to about 97 %, while Morfessor only reaches 62 %, and MORSEL produces the worst results with 20 % at the 100 000 training item mark.

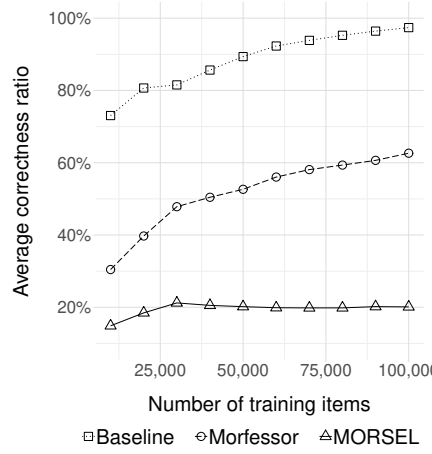
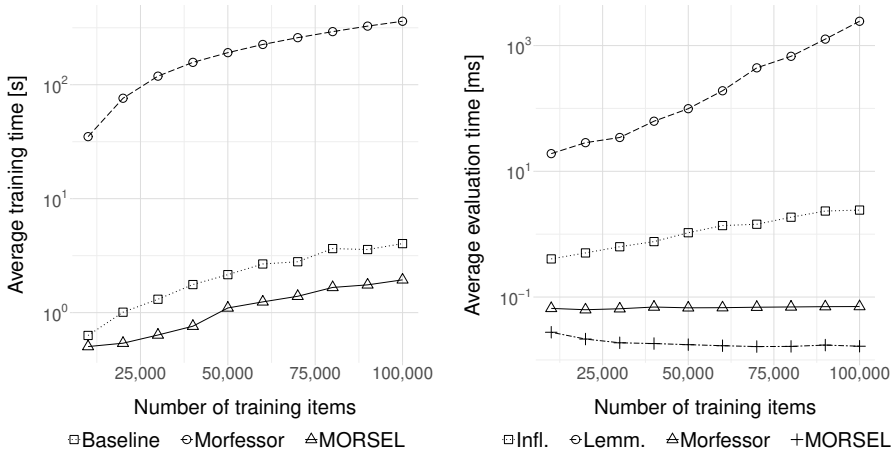


Figure 4. Average correctness ratio of the baseline Morpher model and other morphology engines

Figure 5 a) displays the average training time of the three methods in seconds, using exponential scale on the y axis. Morfessor has the worst training time, since it can learn the segmentation of 10 000 words in 35 seconds, while it takes 6 minutes to learn the segmentation of 100 000 words. MORSEL has a slightly faster training phase and similar characteristics to the baseline Morpher model. The baseline Morpher model trains in 4.03 seconds, while MORSEL finishes learning in 1.94 seconds at the 100 000 training item mark. However, since the baseline Morpher model has a much higher correctness ratio, this is not a big problem.

In Figure 5 b) we can see the average inflection and lemmatization times of the baseline Morpher model, as well as the average segmentation time of Morfessor and MORSEL in milliseconds, using exponential scale on the y axis. Morfessor and MORSEL (the bottom two lines) can perform the segmentation of a word in an average of 70 and 16 microseconds, respectively. This is almost constant time, which lets us draw the conclusion that they work with a map-like structure, identifying the pre-learnt segments in the input words without much searching. Since both inflection and lemmatization are more complex problems, they have a slightly steeper curve (the top two lines). While inflection increases from 0.4 milliseconds to 2.4 milliseconds, lemmatization takes an average of 19.1 milliseconds to 2.4 seconds. This confirms that lemmatization has an exponential time complexity.



a) Average training time

b) Average evaluation time

Figure 5. Average training and evaluation times of the baseline Morpher model and other morphology engines using exponential scale on the y axis

Table 1 contains the average knowledge base size of the baseline Morpher model, Morfessor and Hunmorph-Ocamorph using 100 000 training items. MORSEL does not have an option to export its knowledge base. Morfessor has the smallest database with 3.5 megabytes, but it needs to store much less information. The baseline Morpher model's knowledge base is 6.4 megabytes, but it contains the possible affix type chains, their conditional probabilities, the valid lemmas and more complex transformation rules as well. Hunmorph-Ocamorph has the biggest database with 22.7 megabytes.

Model	**File Size [MB]**
Baseline	6.4
Morfessor	3.5
Hunmorph-Ocamorph	22.7

Table 1. Average knowledge base size of the baseline Morpher model and other morphology engines using 100 000 training items

6.2 Evaluation of the Optimization Techniques

For this evaluation we analyzed the four optimization parameters using a smaller training data set of 100 000 random items to decide which one is worth using with larger data sets.

Figure 6 shows the average correctness ratio on the y axis, and the number of retained atomic rules on the x axis using exponential scale. From this graph we

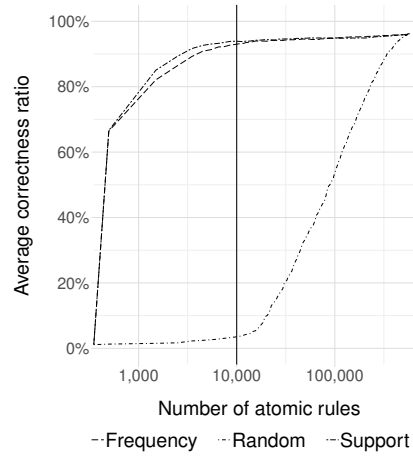
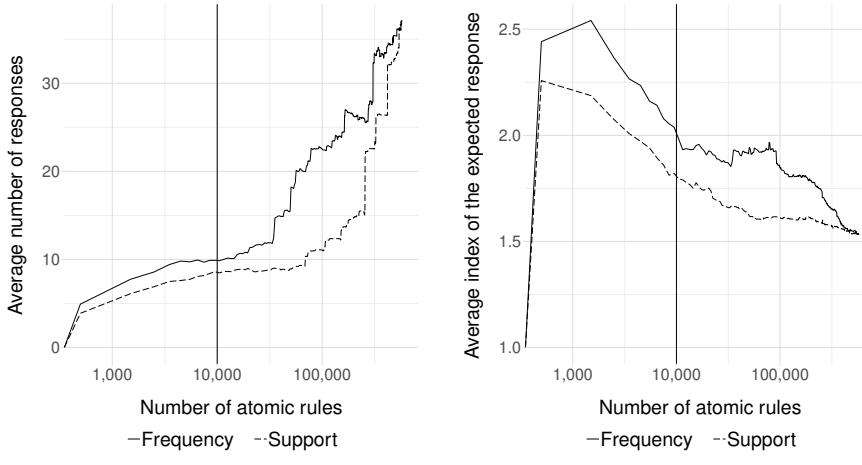


Figure 6. Average correctness ratio over the number of retained atomic rules after p_{supp} and p_{freq} optimization, using exponencial scale on the x axis

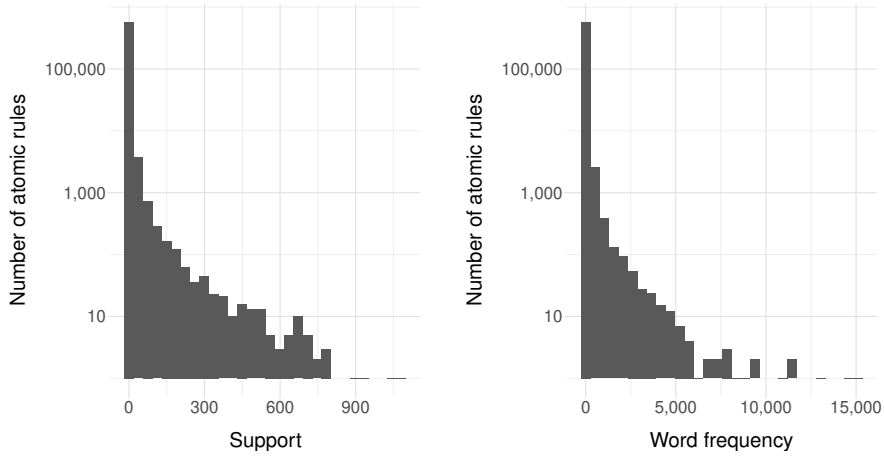
can see that if we drop atomic rewrite rules randomly, the correctness ratio drops dramatically.



a) Average number of responses b) Average index of the expected response

Figure 7. Average number of responses and average index of the expected response over the number of retained atomic rules after p_{supp} and p_{freq} optimization

On the other hand, if we use one of the two optimization parameters, we can reduce the rule base size to about 1.73 % of the original and still inflect and lemmatize about 93 % of the previously unseen words correctly. We added a vertical line to the



a) Support b) Word frequency

Figure 8. Histogram of atomic rules based on their support and word frequency

10 000 atomic rule mark, while the total number of atomic rules was 578 497. We can also see that using p_{supp} we can keep a slightly higher correctness ratio with the same amount of atomic rules, but the two parameters result in very similar results otherwise.

In Figure 7 a) we can see the average number of responses. Similarly to the correctness ratio, p_{supp} performs better, producing slightly fewer responses using the same amount of retained atomic rules.

Figure 7 b) displays the average index of the expected response, i.e., which response is the expected (correct) one. Although it is not guaranteed that the expected response is the correct one due to the large volumes of evaluation data, this metric is a good approximation. The p_{supp} parameter performs better, and the worst value with a small number of retained atomic rules does not increase above 3, meaning that the first 3 responses always contain the expected one.

From the above figures, we can choose $p_{supp} = 10$ as the optimization parameter. The histogram of the atomic rules in Figure 8 shows that choosing a relatively low threshold will drop a lot of rules from the rule base.

Table 2 displays the average number of retained atomic rules, correctness ratio, number of responses and expected response index using different (p_{gen}, p_{max}) combinations. The most responses are produced when we only keep the most general rules, and the correctness ratio is one of the lowest values as well. With $p_{gen} = 3$, the correctness ratio dropped to about 80 %.

p_{gen}	p_{max}	Rules	Correctness [%]	Responses	Response Index
–	–	578 497	96.19	37.11	1.53
1	1	5 019	85.78	126.01	31.94
1	2	16 923	92.62	114.34	7.34
1	3	46 506	94.89	89.45	2.44
1	4	103 533	96.17	56.23	1.60
1	5	175 334	96.18	41.78	1.54
2	1	10 501	91.91	10.92	5.42
2	2	36 186	92.62	9.71	2.05
2	3	90 710	94.02	6.47	1.47
2	4	161 132	94.28	4.60	1.39
2	5	238 879	94.29	4.04	1.39

Table 2. Average number of retained atomic rules, correctness ratio, number of responses and expected response index using different (p_{gen}, p_{max}) combinations

6.3 Evaluating the Optimal Optimization Parameter Using Large Training Data Sets

We wanted to evaluate $(p_{gen} = 1, p_{max} = 1)$, as well as $p_{supp} = 10$ using big training data sets containing up to 3 million training items, but we had to omit the first case, as it could not even handle 500 000 training items due to the exponential growth of responses. Figure 9 shows the average training time of the optimized Morpher model using $p_{supp} = 10$, increasing about linearly up to about 74.61 seconds.

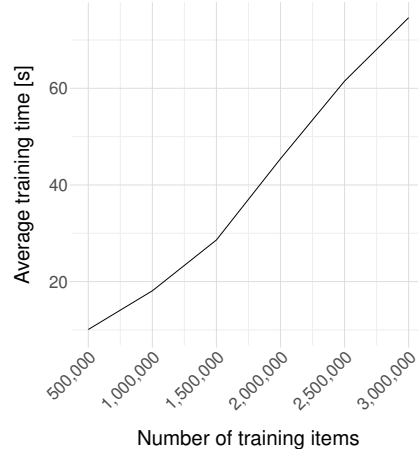
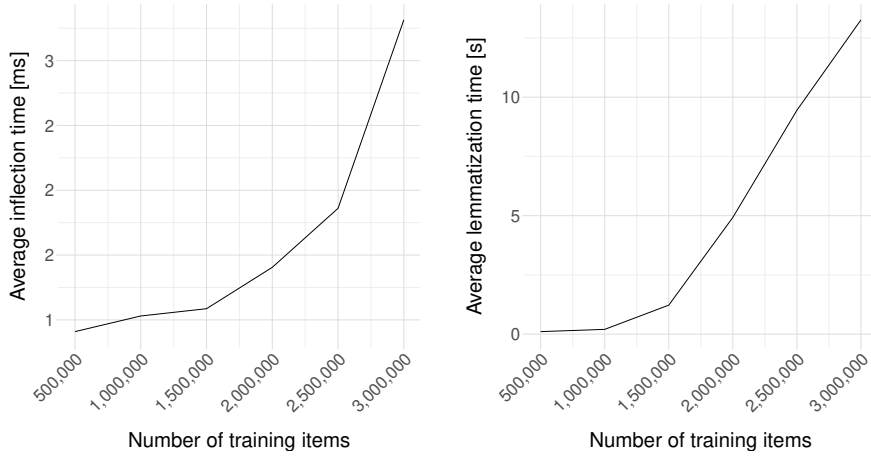


Figure 9. Average training time using big training data sets and $p_{supp} = 10$

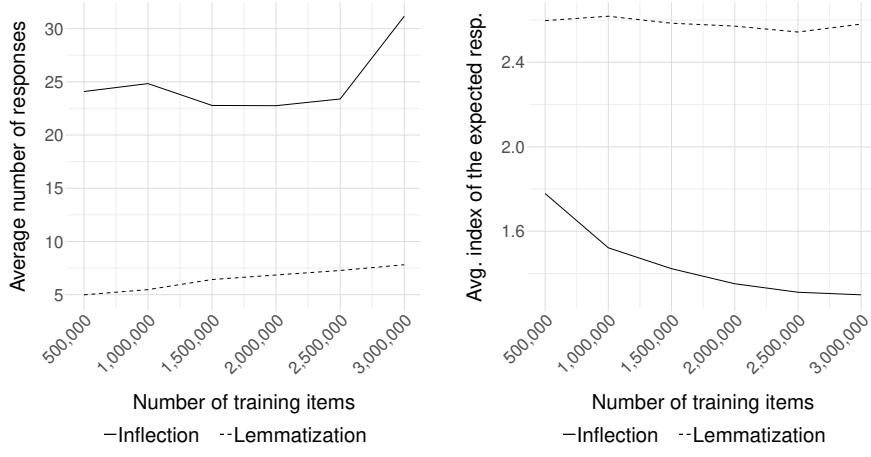
In Figures 10 a) and 10 b) we can see the average inflection and lemmatization times: 3.31 seconds for inflection and 13.26 seconds for lemmatization using 3 million training items.



a) Average inflection time b) Average lemmatization time

Figure 10. Average inflection and lemmatization times using big training data sets and $p_{supp} = 10$

Figure 11 a) displays the average number of responses. It is surprising that inflection produces more responses in average (31.16 vs 7.81).



a) Average number of responses b) Average index of the expected response

Figure 11. Average number of responses and average index of the expected response using big training data sets and $p_{supp} = 10$

The reason is that many lemmatization responses are filtered out due to not ending in a valid lemma. However, Figure 11 b) proves that the index of the expected response does not go above 1.3 and 2.58, respectively.

In Figure 12 we can see the average correctness ratio, that increases from about 96.22 % to about 98.11 % in average.

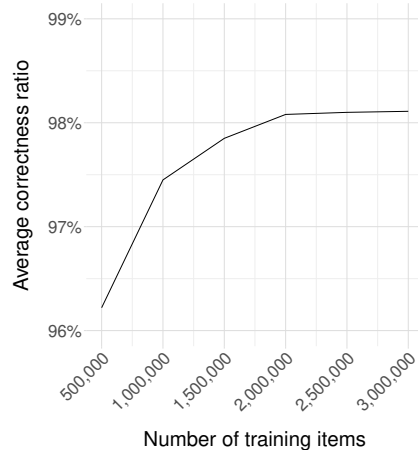


Figure 12. Average correctness ratio using big training data sets and $p_{supp} = 10$

Table 3 contains the summary of the examined metrics, including their baseline values and the measured values for the optimized case ($p_{supp} = 10$), as well as their ratio, using 3 million training items. Just for comparison we executed this test using the baseline Morpher model, but with less evaluation data to save time. The table shows that there are huge improvements, except for the training time: lemmatization of one word would take about 8.5 minutes in average without any optimizations, compared to 13.26 seconds in case of $p_{supp} = 10$, which means that using this optimization parameter value, the lemmatization becomes 2.59 % of the original value.

	Baseline	$p_{supp} = 10$	**Ratio**
Training time [s]	53.41	74.61	139.69 %
Inflection time [s]	640	3.31	0.52 %
Lemmatization time [s]	511.83	13.26	2.59 %
Number of atomic rules	11 354 255	255 867	2.25 %
Knowledge base size [MB]	130.6	5.5	4.21 %

Table 3. Summary of the average values and improvements of the examined metrics using a big training data set containing 3 million items and $p_{supp} = 10$

7 CONCLUSION

In this paper we performed the space and time complexity analysis of the Morpher morphological rule induction model, and introduced several optimization techniques.

The four new optimization parameters aim to reduce the number of retained transformation rules during the training phase. The first optimization parameter (p_{max}) limits the number of generated rules per word pair, while another one (p_{gen}) sets a lower boundary on the context length of the retained rules. We can also reduce the rule base size using statistics calculated from the training data: there is a p_{supp} and a p_{freq} optimization parameter with which we can drop rules that have a support value or a word frequency value less than these threshold parameters.

The complexity analysis showed that these optimization parameters improve the memory requirements and average runtime of the original Morpher model dramatically. The winning configuration was $p_{supp} = 10$ that managed to reduce the number of rules to the 1.73 % of the original set, still keeping an average correctness ratio of about 93 % and finished in acceptable time using up to 3 million training items.

REFERENCES

[1] Bauer, L.: Introducing Linguistic Morphology. Edinburgh University Press, Edinburgh, 2003.

[2] Bergmanis, T.—Goldwater, S.: From Segmentation to Analyses: A Probabilistic Model for Unsupervised Morphology Induction. Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2017), Vol. 1, 2017, pp. 337–346, doi: 10.18653/v1/e17-1032.

[3] Creutz, M.—Lagus, K.: Inducing the Morphological Lexicon of a Natural Language from Unannotated Text. Proceedings of the International and Interdisciplinary Conference on Adaptive Knowledge Representation and Reasoning (AKRR '05), 2005, pp. 106–113.

[4] de la Higuera, C.: Grammatical Inference: Learning Automata and Grammars. Cambridge University Press, 2010, doi: 10.1017/cbo9781139194655.

[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 (EACL '09), 2009, pp. 29–32, doi: 10.3115/1609049.1609057.

[6] Koskenniemi, K.: Two-Level Morphology: A General Computational Model for Word-Form Recognition and Production. Department of General Linguistics, University of Helsinki, Finland, 1983.

[7] Kovács, L.—Szabó, G.: String Transformation Approach for Morpheme Rule Induction. Procedia Technology, Vol. 22, 2016, pp. 854–861, doi: 10.1016/j.protcy.2016.01.060.

[8] Lignos, C.: Learning from Unseen Data. Proceedings of the Morpho Challenge 2010 Workshop, 2010, pp. 35–38.

[9] Jurafsky, D.—Martin, J. H.: Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition. 2nd Edition. Prentice Hall, 2008.

[10] Mohri, M.: Finite-State Transducers in Language and Speech Processing. Computational Linguistics, Vol. 23, 1997, No. 2, pp. 269–311.

[11] Narasimhan, K.—Barzilay, R.—Jaakkola, T.: An Unsupervised Method for Uncovering Morphological Chains. Transactions of the Association for Computational Linguistics, Vol. 3, 2015, pp. 157–167, doi: 10.1162/tacl_a_00130.

[12] Oncina, J.—García, P.—Vidal, E.: Learning Subsequential Transducers for Pattern Recognition Interpretation Tasks. IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 15, 1993, No. 5, pp. 448–458, doi: 10.1109/34.211465.

[13] Pirinen, T.—Lindén, K.: Creating and Weighting Hunspell Dictionaries as Finite-State Automata. Investigationes Linguisticae, Vol. 21, 2010, pp. 1–16, doi: 10.14746/il.2010.21.1.

[14] 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 (ACL '99), 1999, pp. 261–268, doi: 10.3115/1034678.1034723.

[15] Shalonova, K.—Flach, P. A.: Morphology Learning Using Tree of Aligned Suffix Rules. ICML Workshop: Challenges and Applications of Grammar Induction, 2007.

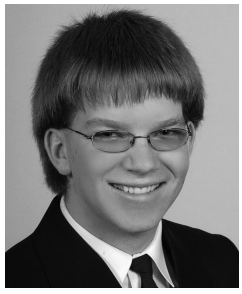
[16] Szabó, G.—Kovács, L.: Benchmarking Morphological Analyzers for the Hungarian Language. Annales Mathematicae et Informaticae, Vol. 49, 2018, pp. 141–166, doi: 10.33039/ami.2018.05.001.

[17] Szabó, G.—Kovács, L.: An Advanced Semi-Supervised Morphological Rule Induction Model for the Hungarian Language. Submitted to a Journal. Available at: `https://users.iit.uni-miskolc.hu/~szabo84/articles/optimization`, 2018.

[18] Trón, V.—Kornai, A.—Gyepesi, Gy.—Németh, L.—Halácsy, P.—Varga, D.: Hunmorph: Open Source Word Analysis. Proceedings of the Workshop on Software (Software '05), 2005, pp. 77–85, doi: 10.3115/1626315.1626321.

[19] 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.

[20] Virpioja, S.—Smit, P.—Grönroos, S.—Kurimo, M.: Morfessor 2.0: Python Implementation and Extensions for Morfessor Baseline. Aalto University, 2013.



Gábor Szabó is currently Ph.D. student at the Institute of Information Technology at the University of Miskolc, Hungary. He has his B.Sc. in software information technology since 2012 and his M.Sc. in engineering information technology since 2014. His main research area is solving the morphological rule induction problem for morphologically complex agglutinative languages.

László Kovács is the Head of the Institute of Information Technology at the University of Miskolc, Hungary, where he is Associate Professor. He received his Ph.D. in computer science in 1998. His broad research areas include morphology, ontologies and database systems, among others.