DOI: 10.14750/ME.2022.020

UNIVERSITY OF MISKOLC

FACULTY OF MECHANICAL ENGINEERING AND INFORMATICS

Pattern Matching Based Automated Learning of Inflection Generation and Morphological Analysis

PHD DISSERTATION

AUTHOR:
Gábor SZABÓ
MSc in Information Engineering

"József Hatvany" DOCTORAL SCHOOL
OF INFORMATION SCIENCE, ENGINEERING AND TECHNOLOGY

Research Area
APPLIED COMPUTATIONAL SCIENCE
Research Group
DATA AND KNOWLEDGE BASES, KNOWLEDGE INTENSIVE SYSTEMS

HEAD OF DOCTORAL SCHOOL:
Prof. Dr. Jenő SZIGETI

ACADEMIC SUPERVISOR:
Prof. Dr. László KOVÁCS

Miskolc
2021

DOI: 10.14750/ME.2022.020

Declaration of Authorship

The author hereby declares that this thesis has not been submitted, either in the same or in different form, to this or to any other university for obtaining PhD degree.

The author confirms that the submitted work is his own and the appropriate credit has been given where reference has been made to the work of others.

Szerzői nyilatkozat

Alulírott Szabó Gábor kijelentem, hogy ezt a doktori értekezést magam készítettem, és abban csak a megadott forrásokat használtam fel.

Minden olyan részt, amelyet szó szerint vagy azonos tartalomban, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Miskolc, 2021. november 2.

Szabó Gábor

A disszertáció bírálatai és a védésről készült jegyzőkönyv megtekinthető a Miskolci Egyetem Gépészmérnöki és Informatikai Karának Dékáni hivatalában, valamint a doktori iskola weboldalán: `http://www.hjphd.iit.uni-miskolc.hu`

DOI: 10.14750/ME.2022.020

Recommendation

Our professional collaboration with my PhD student Gábor Szabó started a long time ago. He first got involved in the work at the Institute of Information Science during his MSc studies, as a demonstrator and by writing research papers for the *Scientific Students' Associations* (*"TDK"*) conference. He was admitted to the *"József Hatvany"* Doctoral School of Information Science, Engineering and Technology in 2014, since then we have been working closely as supervisor and PhD student.

Based on the experience gained during our long scientific relationship, Gábor Szabó stands out from other students in many ways. His main strengths are:

- demanding and precise work,
- pursuing thoroughly tested solutions,
- strong background knowledge in information technology and professional experience in software engineering,
- exceptional reliability and
- strong English language skills.

Gábor Szabó has successfully fulfilled the requirements of the Doctoral School to acquire the pre-degree certificate, and he has also been evolving his publication skills gradually and deliberately over the years. It is worth mentioning that in the field of his research domain (machine learning processes, optimization and application in the area of natural language processing), he managed to publish several quality research papers, including

- 2 papers in journals with impact factor,
- 1 paper in a Q2 journal,
- 2 papers in Q3 journals and
- 2 papers in Q4 journals.

In his doctoral thesis, he managed to strike a balance between theory and practice, by creating reference implementations and test systems to support the results of his theoretical analysis. The preliminary defence, that was attended by both domestic and foreign reviewers, also confirmed the outstanding scientific results of Gábor Szabó.

Based on the candidate's results and attitude, I support the public defence of his doctoral thesis.

Miskolc, November 8, 2021

Prof. Dr. László KOVÁCS
academic supervisor

DOI: 10.14750/ME.2022.020

Summary

This dissertation focuses on the pattern matching based automated learning of inflection generation and morphological analysis. The problem is approached on two levels: first, single-affix morphology models are proposed that can learn transformation rules from a training word pair set related to a single affix type, then a higher-level morphology model is proposed that can manage multiple affix types, too. The proposed models are evaluated against Hungarian, a morphologically complex agglutinative language.

There are several other existing morphology models that can be used for the Hungarian language. The first step of this research project was to analyze four of the most popular such models, including Hunmorph-Ocamorph, Hunmorph-Foma, Humor and Hunspell. According to the experimental results relying on formal measures, the most usable morphology model among them is Hunmorph-Ocamorph.

The first proposed single-affix transformation engine model has a complex rule structure that contains position indices and describes not only the context of the transformation, but also its elementary transformation steps. The generated rules are stored in a compact lattice structure that can be built using three different builder algorithms. The other single-affix model is called ASTRA (Atomic String Transformation Rule Assembler), and has a simplified rule model that omits position indices and describes the transformations as string replacement operations. Evaluation shows that while the lattice based model achieves a more compact storage structure, the ASTRA model has an exceptional accuracy and generalization capability, beating the base models such as simple dictionaries, FSTs and the TASR model.

For multi-affix inflection generation and morphological analysis learning, I propose the Morpher model that can manage multiple affix types. During the training phase, Morpher deduces the training word pair sets based on a more generic training data set, and builds a separate ASTRA instance for each affix type of the target language. The model also calculates the conditional probabilities of the valid affix type chains, and stores the valid lemmas and their possible parts of speech. The experimental results show that Morpher's average accuracy beats all the examined base models including 6 SIGMORPHON models, 3 unsupervised segmentation models and 2 analyzer models, while achieving low average training, inflection and analysis time.

After performing the space and time complexity analysis of the Morpher and ASTRA models, I propose 3 optimization techniques that aim to reduce the rule base size and thus the average inflection and analysis time. The winner optimization technique eliminates candidate atomic rules during the training phase by using their support values. After applying this optimization technique, the Morpher model became able to perform inflection generation and morphological analysis in acceptable finite time after being trained using up to 3 million training items. Comparison shows that the average training, inflection and analysis times are reduced dramatically, while keeping the average accuracy high.

The reference implementation of the proposed models can be found in a few Github projects. As of writing, these projects are implemented in Java 17, utilizing its module system and parallel streams. The built binaries are published on jcenter and Maven Central. For ecosystems other than Java, a Spring Boot based server-side Morpher REST API application is published as well, in the form of a Docker image. This API is consumed by a React and React Native based client application, capable of running in the browser, as well as on Android and iOS devices.

DOI: 10.14750/ME.2022.020

Összefoglalás

A disszertáció témája a ragozás és morfológiai elemezés mintaillesztésen alapuló automatizált tanulása. A problémát két szinten közelítem meg: először két olyan modellt mutatok be, amely képes egy tanító szópárhalmazból megtanulni egyetlen toldaléktípus transzformációit, majd egy magasabb szintű modellt írok le, amely képes egyszerre több toldaléktípust is kezelni. A modelleket magyar nyelvű adatok segítségével tesztelem, amely egy morfológiailag komplex, agglutináló nyelv.

Az irodalomban jelenleg is található több olyan morfológiai modell, amely támogatja a magyar nyelvet. A kutatási projekt első lépése az volt, hogy kielemeztem négy népszerű modellt ezek közül, beleértve a Hunmorph-Ocamorph, Hunmorph-Foma, Humor és Hunspell eszközöket. A formális mérőszámokon alapuló eredmények szerint a leginkább használható modell ezek közül a Hunmorph-Ocamorph.

Az egytoldalékos eset első bemutatott modellje egy hálóalapú modell, amely a generált szabályokat egy tömör hálóban tárolja, melynek felépítéséhez három bemutatott hálóépítő algoritmus is használható. A szabályok struktúrája viszonylag komplex, mivel pozícióindexeket és a traszformáció elemi lépéseit is tartalmazza. A másik modell neve ASTRA (az angol *Atomic String Transformation Rule Assembler* rövidítése). Ennek a modellnek a jellemzője, hogy a szabályleírója jóval egyszerűbb, elhagyja a pozícióindexeket, a transzformációkat pedig egyszerű string csereként modellezi. A modellek kiértékeléséből látszik, hogy míg a hálóalapú modell tömörebb struktúrához vezet, az ASTRA modell kivételesen jó pontosságot képes elérni, összehasonlítva az egyszerű szótárakkal, FST-kkel és a TASR modellel.

A többtoldalékos eset megoldására a Morpher nevű modellt készítettem el. Betanítás közben a Morpher modell minden egyes toldaléktípushoz egy saját, különálló ASTRA példányt épít, melyet egy általánosabb tanítóhalmazból generált szópárhalmazzal tanít be. Emellett kiszámolja a talált toldaléktípus láncok feltételes valószínűségeit, és eltárolja a nyelv lemmáit, valamint azok lehetséges szófajait. A teszteredményekből látszik, hogy a Morpher átlagos pontosságban felülmúlja az összes vizsgált alapmodellt, köztük 6 SIGMORPHON modellt, 3 szegmentációs modellt és 2 elemző modellt, alacsony betanítási, ragozási és elemzési idő mellett.

A Morpher és ASTRA modellek hely- és időkomplexitásának vizsgálata után 3 optimalizációs technikát mutatok be, amelyek célja, hogy a tudásbázis méretének csökkentésével a modellek időigényét is csökkentsék. A nyertes optimalizáció a szabályjelöltek *support* értékei alapján dob el szabályokat betanítás közben. Ezt a technikát alkalmazva a Morpher modell képessé válik akár 3 millió tanító mintával történő betanítás után is elfogadható, véges időben elvégezni a ragozás és elemzés műveletét, amire optimalizáció nélkül nem lenne képes. Kisebb tanítóméretek esetén összehasonlítva az időeredményeket, látszik a drámai javulás, miközben az átlagos pontosság továbbra is magas marad.

A bemutatott modellek referenciaimplementációi forráskóddal együtt megtalálhatóak különböző Github projektekben. A Morpher keretrendszert (beleértve a hálóalapú modellt és az ASTRA-t is), Java-ban fejlesztettem, kihasználva annak modulrendszerét és párhuzamos, funkcionális adatfeldolgozó elemeit. A keletkezett JAR fájlokat a jcenter és Maven Central repositorykban publikálom. A Javán kívüli ökoszisztémák számára egy Spring Boot alapú, szerver oldali Morpher REST API alkalmazást is fejlesztettem, amelyből egy Docker image publikálódik. Az API köré pedig egy React és React Native alapú kliens alkalmazást készítettem, amely képes futni böngészőben, illetve Android és iOS eszközökön egyaránt.

DOI: 10.14750/ME.2022.020

Acknowledgements

Firstly, I would like to express my gratitude to my academic supervisor, Prof. Dr. László Kovács, for his continuous support since my BSc studies. Without his help and insight, this work would not have been possible.

I would also like to thank my family, especially my Mom and Dad, for helping me mentally and humanly, and supporting me with infinite patience, during good times and bad times. I know that without your support and attitude, not only would I be in a completely different situation, but I would also be a different person altogether.

DOI: 10.14750/ME.2022.020

Contents

Declaration of Authorship i

Recommendation ii

Summary iii

Összefoglalás iv

Acknowledgements v

Contents vi

List of Notations x

List of Abbreviations xiv

List of Figures xv

List of Tables xvii

1 Introduction 1
- 1.1 Problem domain . . . 1
- 1.2 Natural Language Categorization . . . 3
  - 1.2.1 Historical Language Families . . . 3
  - 1.2.2 Linguistic Categories . . . 3
  - 1.2.3 Morphological Language Classes . . . 4
- 1.3 Research Goals . . . 5
- 1.4 Dissertation Guide . . . 6

2 Survey of the Current Models 8
- 2.1 Categorization of the Existing Morphology Models . . . 8
  - 2.1.1 Knowledge Representation . . . 8
    - Dictionary Based Systems . . . 9
    - Rule Based Systems . . . 10
    - Statistical Methods . . . 11
    - Artificial Intelligence Based Methods . . . 12
  - 2.1.2 Scope . . . 12
    - Single-Affix Models . . . 12
    - Multi-Affix Models . . . 13
  - 2.1.3 Symmetry . . . 13
    - Asymmetric Models . . . 13
    - Symmetric Models . . . 14
  - 2.1.4 Granularity of Analysis . . . 14
    - Morphological Analysis . . . 14

DOI: 10.14750/ME.2022.020

Segmentation . . . 14
Lemmatization . . . 14
Stemming . . . 15
2.1.5 Machine Learning Capabilities . . . 15
Non-Automated Training . . . 15
Supervised Training . . . 15
Unsupervised Training . . . 16
Semi-Supervised Training . . . 17
2.2 Main Baseline Morphology Models . . . 18
2.2.1 Two-Level Morphology . . . 18
2.2.2 Finite State Transducer (FST) . . . 19
2.2.3 Tree of Aligned Suffix Rules (TASR) . . . 21
2.2.4 Unsupervised Segmentation Models . . . 22
Morfessor 2.0 . . . 22
MORSEL . . . 23
MorphoChain . . . 24
2.2.5 SIGMORPHON . . . 24
SIGMORPHON 2016 . . . 25
SIGMORPHON 2017 . . . 25
SIGMORPHON 2018 . . . 25
2.2.6 Morphological Analyzers for the Hungarian Language . . . 26
Hunmorph-Ocamorph . . . 26
Hunmorph-Foma . . . 26
Humor . . . 27
Hunspell . . . 27
2.3 Conclusion . . . 28

3 The Analysis of Existing Hungarian Morphological Analyzers 30
3.1 Similarity and Distance of Morphological Analyzers . . . 31
3.2 Analyzing the Similarities and Differences of the Morphological Analyzers . . . 32
3.2.1 Comparison of the Annotation Token Systems . . . 33
3.2.2 Recognition Statistics . . . 33
3.2.3 Mapping Among the Examined Morphological Analyzers . . . 36
3.2.4 Cumulative Distance . . . 37
3.3 Conclusion . . . 38

4 Single-Affix Transformation Engine Model 39
4.1 Lattice Based Model . . . 39
4.1.1 The Theory of Formal Concept Analysis . . . 40
4.1.2 Levenshtein Distance Based Transformation Rule Generation . . . 41
Unit Cost Model for Levenshtein Distance Calculation . . . 41
Improved Cost Function . . . 42
4.1.3 The Lattice Rule Model . . . 44
4.1.4 Lattice Builder Algorithms . . . 46
Complete Lattice Builder . . . 46
Consistent Lattice Builder . . . 47
Minimal Lattice Builder . . . 48
4.1.5 Inflection Generation . . . 49
4.2 Atomic String Transformation Rule Assembler (ASTRA) . . . 49
4.2.1 The ASTRA Rule Model . . . 50

DOI: 10.14750/ME.2022.020

4.2.2 The Training Method of ASTRA . . . 50
4.2.3 Inflection Generation . . . 53
4.2.4 Morphological Analysis . . . 54
4.3 Experiments . . . 54
4.3.1 Average Training Time . . . 55
4.3.2 Average Size . . . 56
4.3.3 Average Search Time . . . 56
4.3.4 Average Accuracy . . . 57
4.4 Conclusion . . . 58

5 Multi-Affix Morphology Model **61**
5.1 Architecture of the Proposed Model . . . 61
5.2 The Formal Model of Concatenative Morphology . . . 63
5.3 The Training Phase of Morpher . . . 64
5.4 Performing Inflection Generation Using Morpher . . . 66
5.5 Performing Morphological Analysis Using Morpher . . . 67
5.6 Experimental Results . . . 69
5.6.1 Average Training Time . . . 70
5.6.2 Average Size . . . 71
5.6.3 Average Inflection and Analysis Time . . . 71
5.6.4 Average Accuracy . . . 72
5.6.5 Generalization Capabilities . . . 74
5.6.6 Cross-Validation with the SIGMORPHON Data Sets . . . 74
5.7 Conclusion . . . 75

6 Complexity Analysis and Optimization of Morpher and ASTRA **77**
6.1 Complexity Analysis . . . 77
6.1.1 Space Complexity . . . 77
6.1.2 Time Complexity . . . 79
6.2 Optimization Techniques . . . 80
6.2.1 Eliminating the Redundant Atomic Rules . . . 80
6.2.2 Limiting the Generalization Factor . . . 81
6.2.3 Indirect Noise Reduction . . . 82
6.3 Empirical Analysis of the Optimization Parameters . . . 82
6.4 Evaluation . . . 85
6.4.1 Comparison with the Baseline Morpher Model . . . 85
Average Training Time . . . 85
Average Size . . . 85
Average Inflection and Analysis Time . . . 87
Average Accuracy . . . 88
6.4.2 Using Big Training Data Volumes . . . 88
Average Training Time . . . 88
Average Size . . . 89
Average Inflection and Analysis Time . . . 89
Average Accuracy . . . 90
6.5 Conclusion . . . 91

DOI: 10.14750/ME.2022.020

7 The Reference Implementation of the Morpher Ecosystem **94**
7.1 The Training and Evaluation Data Generation Process 94
7.2 The Layers of the Morpher Ecosystem 96
7.2.1 Morpher Framework 96
Morpher Core 97
Morpher Transformation Engines 97
Morpher Language Handlers 98
Morpher Engines 98
7.2.2 Morpher API 100
7.2.3 Morpher Client 101
7.3 Conclusion 102

8 Conclusion **104**
8.1 Contribution 104
8.2 Future work 106

A Mapping of the Examined Annotation Token Systems **107**

Author's Publications **119**

References **121**

DOI: 10.14750/ME.2022.020

List of Notations

General notations

$\{x_i\}_{i=1}^n$ An unordered set of n items $x_1, \ldots, x_n$
$\langle x_i \rangle_{i=1}^n$ An ordered list of n items $x_1, \ldots, x_n$
Σ The alphabet containing non-empty characters
Σ^k The set of strings with a length of k above the Σ alphabet
Σ^* The set of all the strings above Σ, i.e. $\cup_{i=0}^{\infty} \Sigma^i$
c An arbitrary character in the Σ alphabet
$\emptyset$ The empty character
s An arbitrary string in Σ^*
ϵ The empty string
s_i The ith character in the string s
$|s|$ The length of the string s
s^{-1} The reverse of the string s, i.e. if $s = s_1 \ldots s_k$ then $s^{-1} = s_k \ldots s_1$
W The set of meaningful words in the target language. $W \subset \Sigma^*$
w An arbitrary meaningful word in W
$w\left[i, j\right]$ The substring of the word w from the ith character to the jth character
(w_l, w_r) An arbitrary word pair from the words in W
$\mathbb{I}$ The set of training word pairs for single-affix transformation engines
$\bar{W}$ The set of lemmas in the target language. $\bar{W} \subset W$
$\bar{w}$ An arbitrary lemma in $\bar{W}$
$\mathbb{T}$ The set of affix types in the target language
T An arbitrary affix type in $\mathbb{T}$
$\bar{\mathbb{T}}$ The set of parts of speech in the target language
$\bar{T}$ An arbitrary part of speech in $\bar{\mathbb{T}}$
$T_i \rightarrow T_j$ The affix type T_j can be applied after T_i
$\bar{w} \Rightarrow w$ The word w is reachable from the lemma $\bar{w}$

Analysis of existing Hungarian morphological analyzers

A An arbitrary morphological analyzer
$\mathbb{T}^A$ The set of annotation tokens related to the morphological analyzer A
T^A An arbitrary annotation token related to the morphological analyzer A
$l\left(A\left(w\right)\right)$ The lemma projection of the morphological analyzer relation
$t\left(A\left(w\right)\right)$ The annotation token list projection of the morphological analyzer relation
W^A The set of recognized words by the morphological analyzer A
ν^A The ratio of W^A in the whole corpus
$S^R_{A_i,A_j}$ The recognition similarity metric of the morphological analyzers A_i and A_j
$D^R_{A_i,A_j}$ The recognition distance metric of the morphological analyzers A_i and A_j
m_{A_i,A_j} The mapping function of the annotion token systems of A_i and A_j
$S^T_{A_i,A_j,m_{A_i,A_j}}$ The token similarity metric of the morphological analyzers A_i and A_j based on the mapping m_{A_i,A_j}
$D^T_{A_i,A_j,m_{A_i,A_j}}$ The token distance metric of the morphological analyzers A_i and A_j based on the mapping m_{A_i,A_j}

DOI: 10.14750/ME.2022.020

$S^M_{A_i,A_j,m_{A_i,A_j}}$ The mapping similarity metric of the morphological analyzers A_i and A_j based on the mapping m_{A_i,A_j}
$D^M_{A_i,A_j,m_{A_i,A_j}}$ The mapping distance metric of the morphological analyzers A_i and A_j based on the mapping m_{A_i,A_j}
$D^C_{A_i,A_j,m_{A_i,A_j}}$ The cumulative distance metric of the morphological analyzers A_i and A_j based on the mapping m_{A_i,A_j}

Lattice based model

R^L An arbitrary transformation rule of the lattice based transformation engine model. $R^L = \left(\alpha^L, \sigma^L, \omega^L, \eta^L_f, \eta^L_b, \Delta^L\right)$
α^L The prefix component of a transformation rule, i.e. some characters before the changing substring in the base word form
σ^L The core component of a transformation rule, i.e. the changing substring of the base word form
ω^L The postfix component of a transformation rule, i.e. some characters after the changing substring in the base word form
η^L_f The front index of the rule context occurrence in the source word from its beginning
η^L_b The back index of the rule context occurrence in the source word from its end
Δ^L The transformation path component of a transformation rule, i.e. the list of elementary transformation steps (character additions, removals, replacements and invariant replacements): $\Delta^L = \left\langle \delta^L_i \right\rangle$
δ^L An elementary transformation step on a transformation path Δ^L. It can be a character addition, removal, replacement or an invariant replacement.
δ^L_+ An arbitrary character addition transformation step
δ^L_- An arbitrary character removal transformation step
$\delta^L_=$ An arbitrary invariant character replacement transformation step
$\delta^L_{\neq}$ An arbitrary character replacement transformation step
$cost\left(\delta^L\right)$ The cost of the elementary transformation step δ^L
$\gamma\left(R^L\right)$ The context of the transformation rule R^L, i.e. the concatenation of its prefix, core and postfix components: $\gamma\left(R^L\right) = \alpha^L + \sigma^L + \omega^L$
$\cap_{\leftarrow}$ The intersection operator for the prefix component of the lattice rule model. This operator intersects the characters from the right side of the substrings until these intersections can be performed.
$\cap_{\leftrightarrow}$ The intersection operator for the core and transformation path components of the lattice rule model. This operator performs a complete intersection of the given substrings and transformation lists.
$\cap_{\rightarrow}$ The intersection operator for the postfix component of the lattice rule model. This operator intersects the characters from the left side of the substrings until these intersections can be performed.
$\bar{\cap}$ The intersection operator for the front and back index component of the lattice rule model. This operator intersects the given indices only if they are equal.

ASTRA model

\$ A special character that marks the start of a word. This character is not part of the original Σ alphabet: $\$ \notin \Sigma$
$\#$ A special character that marks the end of a word. This character is not part of the original Σ alphabet: $\# \notin \Sigma$
$\breve{\Sigma}$ The extended alphabet that contains all the non-special, non-empty characters, and the special \$ and # characters: $\breve{\Sigma} = \Sigma \cup \{\$, \#\}$

DOI: 10.14750/ME.2022.020

$\breve{W}$ The set of meaningful words in the target language, extended with the special \$ and # characters: $\breve{W} = \{\breve{w} \mid \breve{w} = \$ + w + \# \text{ and } w \in W\}$
$\breve{w}$ An arbitrary extended word in $\breve{W}$
μ Operator that extends the input word w with the \$ and # characters. If the input word is $w = w_1 \dots w_k$ then $\mu(w) = \$w_1 \dots w_k\#$
μ^{-1} Operator that removes the \$ and # characters from the input word $\breve{w}$. If the input word is $\breve{w} = \$w_1 \dots w_k\#$ then $\mu^{-1}(\breve{w}) = w_1 \dots w_k$
ψ_l^i The ith segment component in the left word of a word pair
ψ_r^i The ith segment component in the right word of a word pair
R^A An arbitrary atomic rule of the ASTRA transformation engine model. $R^A = (\alpha^A, \sigma^A, \tau^A, \omega^A)$
α^A The prefix component of an atomic rule, i.e. some characters before the changing substring in the base word form
σ^A The changing substring component of an atomic rule, i.e. the characters that need to be replaced in the base word form
τ^A The replacement string component of an atomic rule, i.e. the characters that need to replace σ^A in the base word form
ω^A The postfix component of an atomic rule, i.e. some characters after the changing substring in the base word form
$\gamma(R^A)$ The context of the atomic rule R^A, i.e. the concatenation of its prefix, changing substring and postfix components: $\gamma(R^A) = \alpha^A + \sigma^A + \omega^A$
Γ^A A rule group containing atomic rules with the same context
$\gamma(\Gamma^A)$ The context of the rule group Γ^A
$f(R^A \mid \breve{w})$ The fitness function that calculates the fitness value of the R^A atomic rule for the extended word $\breve{w}$
θ The function that returns how similar the context of an atomic rule is to the input word

Morpher model

λ Maps a word to its possible lemmas. $\lambda : W \to 2^{\bar{W}}$
$\mathfrak{L}$ Maps a lemma to its possible parts of speech. $\mathfrak{L} : \bar{W} \to 2^{\bar{\mathbb{T}}}$
φ Maps a word to its possible affix type lists. $\varphi : W \to \{\langle T_1, \dots, T_k \rangle\}$
$P(\bar{T})$ The probability of the part of speech $\bar{T}$
$\mathfrak{M}$ Function that can determine the conditional probability of an affix type chain
$\mathfrak{M}^{-1}$ Function that can determine the conditional probability of a reversed affix type chain
E_T The transformation engine instance for the affix type T
$\mathcal{FC}^{E_T}$ Operator that converts the given input word to a set of output words using the transformation engine E_T
$\mathcal{BC}^{E_T}$ Operator that converts back the given input word to a set of output words using the transformation engine E_T
$\mathcal{I}$ Operator that performs inflection on an input lemma using a set of affix types
$\mathcal{A}$ Operator that performs morphological analysis on an input inflected word form
$\mathfrak{T}$ The training data of the Morpher model. $\mathfrak{T} = \{(w, \bar{w}, \bar{T}, \langle T_i \rangle)\}$
ϑ_i The aggregated weight of the ith Morpher response

Cost analysis and optimization of ASTRA and Morpher

Θ Encloses the function from above and below. $f(n)$ is $\Theta(g(n))$ if there exist positive constants c_1, c_2 and n_0 such that $0 \le c_1 \cdot g(n) \le f(n) \le c_2 \cdot g(n)$ for all $n \ge n_0$
O Represents the upper bound of a function. $f(n)$ is $O(g(n))$ if there exist positive constants c and n_0 such that $0 \le f(n) \le c \cdot g(n)$ for all $n \ge n_0$

DOI: 10.14750/ME.2022.020

$W^2_{E_T}$ The set of deduced training word pairs for the transformation engine E_T
p_{max} The number of atomic rules to generate at most for each deduced training word pair
p_{gen} The minimum context length of the retained atomic rules
$\Upsilon_{p_{gen}}$ The minimum number of generated atomic rules that have a context shorter than p_{gen}
p_{supp} The minimum support value of the retained atomic rules
p_{freq} The minimum word frequency of the retained atomic rules

DOI: 10.14750/ME.2022.020

List of Abbreviations

AI Artificial Intelligence

CRS Controlled Rewrite System

LMS Labeled Morphological Segmentation

MDL Minimum Description Length
Models
- **ASTRA** Atomic String Transformation Rule Assembler
- **CRF** Conditional Random Field
- **FCA** Formal Concept Analysis
- **FSA** Finite State Automaton
- **FST** Finite State Transducer
- **GA** Genetic Algorithm
- **HMM** Hidden Markov Model
- **MEMM** Maximum Entropy Markov Model
- **NLM** Neural Language Model
- **NN** Neural Network
- **OSTIA** Onward Subsequential Transducer Inference Algorithm
- **PHMM** Pair Hidden Markov Model
- **RNN** Recursive Neural Network
- **SVM** Support Vector Machine
- **TASR** Tree of Aligned Suffix Rules

NLP Natural Language Processing

POS Part of Speech

SOV Subject-Object-Verb
SVO Subject-Verb-Object

DOI: 10.14750/ME.2022.020

List of Figures

1.1 The Indo-European language family . . . 3
1.2 The main components of the proposed morphology model . . . 5

2.1 A simplified view of dictionary based systems . . . 9
2.2 A simplified view of rule based systems . . . 10
2.3 A sample finite state automaton . . . 20
2.4 A sample onward subsequential transducer . . . 20
2.5 A sample tree of aligned suffix rules . . . 21

3.1 Visualizing the annotation token system distances . . . 33
3.2 Venn diagram of the number of recognized words for the three strongest morphological analyzers . . . 35
3.3 Visualizing the recognition distances . . . 36
3.4 Visualizing the mapping distances . . . 37
3.5 Visualizing the cumulative distances . . . 38

4.1 A sample lattice . . . 40
4.2 Two possible Levenshtein matrices with optimal cost for the Hungarian word pair *(alma, almát)* . . . 43
4.3 The structure of a sample lattice . . . 47
4.4 A sample maximal consistent node . . . 49
4.5 The average training time of the single-affix transformation engine models . . . 55
4.6 The average size of the single-affix transformation engine models . . . 56
4.7 The average search time of the single-affix transformation engine models . . . 57
4.8 The average accuracy of the single-affix transformation engine models using subsets of the evaluation word pair set for training . . . 58
4.9 The average accuracy of the single-affix transformation engine models using disjoint training and evaluation word pair sets . . . 59

5.1 The main components of the Morpher model . . . 62
5.2 Multiple lemmas and inflected word forms of the same Hungarian word 64
5.3 The average training time of the multi-affix morphology models . . . 70
5.4 The average evaluation time of the multi-affix morphology models . . 72
5.5 The average accuracy of the multi-affix morphology models . . . 73

6.1 The average accuracy based on the number of retained atomic rules using p_{supp} and p_{freq} optimization . . . 83
6.2 The average number of responses and the average index of the expected response based on the number of retained atomic rules using p_{supp} and p_{freq} optimization . . . 84
6.3 The histogram of the number of atomic rules based on their support values and word frequencies . . . 84

DOI: 10.14750/ME.2022.020

6.4 The average training time of the baseline and the optimized Morpher model . . . 86
6.5 The average size of the baseline and the optimized Morpher model . . 86
6.6 The average inflection time and the average analysis time of the baseline and the optimized Morpher model . . . 87
6.7 The average accuracy of the baseline and the optimized Morpher model 88
6.8 The average training time of the optimized Morpher model using big training data volumes . . . 89
6.9 The average number of atomic rules of the optimized Morpher model using big training data volumes . . . 89
6.10 The average inflection and analysis time of the optimized Morpher model using big training data volumes . . . 90
6.11 The average accuracy of the optimized Morpher model using big training data volumes . . . 90
6.12 The average number of responses and the average index of the expected response of the optimized Morpher model using big training data volumes . . . 91

7.1 The visualization of the valid affix type chains in the Hungarian language based on the generated data set . . . 95
7.2 The architecture of the Morpher ecosystem . . . 97
7.3 The Morpher web client . . . 100
7.4 The Morpher mobile client . . . 101

DOI: 10.14750/ME.2022.020

List of Tables

1.1 The accusative case of some Hungarian words . . . 6

2.1 Categorization of the baseline morphology models . . . 28

3.1 Token similarity $\left(S^T_{A_i,A_j,m_{A_i,A_j}}\right)$ and distance $\left(D^T_{A_i,A_j,m_{A_i,A_j}}\right)$ values among the morphological analyzers . . . 33
3.2 The number of recognized words $\left(\left|W^{A_i}\right|\right)$ and the ratio of recognized words $\left(\nu^{A_i}\right)$ for each morphological analyzer . . . 34
3.3 The number of words recognized by only one morphological analyzer 34
3.4 The number of words recognized by exactly two morphological analyzers . . . 34
3.5 The number of words recognized by exactly three morphological analyzers . . . 35
3.6 Recognition similarity $\left(S^R_{A_i,A_j}\right)$ and distance $\left(D^R_{A_i,A_j}\right)$ values among the morphological analyzers . . . 36
3.7 Mapping similarity $\left(S^M_{A_i,A_j,m_{A_i,A_j}}\right)$ and distance $\left(D^M_{A_i,A_j,m_{A_i,A_j}}\right)$ values among the morphological analyzers . . . 36
3.8 Cumulative distance $\left(D^C_{A_i,A_j,m_{A_i,A_j}}\right)$ values among the morphological analyzers . . . 37

4.1 Hungarian vowel attributes . . . 43
4.2 Hungarian consonant attributes . . . 44
4.3 Sample lattice rules for the artificial word pair *(xabyx**ab**yz, xabyx**cdw**yz)* 45
4.4 Sample rule intersection . . . 46
4.5 Sample atomic rules for the artificial word pair *(xabyx**ab**yz, xabyx**cdw**yz)* 50
4.6 Sample segment decomposition for the word pair *(dob, ledobott)* . . . 51
4.7 Summary of the measured metrics using 10,000 training word pairs and a disjoint evaluation word pair set . . . 60

5.1 The average file size of the exported knowledge bases . . . 71
5.2 The accuracy of the multi-affix morphology models using 100 artificial words imitating the inflection rules of Hungarian accusative case . . . 74
5.3 The accuracy of the Morpher model using the data sets provided by SIGMORPHON . . . 75
5.4 Summary of the measured metrics using 100,000 training items . . . 76

6.1 The average number of retained atomic rules, correctness ratio, number of responses and expected response index using different $(p_{gen},\ p_{max})$ combinations . . . 85
6.2 Summary of the measured metrics of the baseline and the optimized Morpher model using 100,000 training items . . . 92

DOI: 10.14750/ME.2022.020

6.3 Summary of the measured metrics of the optimized Morpher model using 1 million training items . . . 92

A.1 POS categories . . . 107
A.2 Noun features . . . 107
A.3 Verb features . . . 109
A.4 Noun derivations . . . 114
A.5 Verb derivations . . . 115
A.6 Adjective derivations . . . 116
A.7 Numeral derivations . . . 116
A.8 POSTP categories . . . 117

DOI: 10.14750/ME.2022.020

Dedicated to my beloved family…

DOI: 10.14750/ME.2022.020

Chapter 1

Introduction

Nowadays there are several popular research areas in the field of computer linguistics, including natural language processing (NLP), syntactic analysis or the automated learning of morphology. Since these research areas are related to different levels of grammar, it is not uncommon that the results of one such research project can be used as the input of a higher-level problem. The ultimate goal is to be able to process free texts in order to extract the knowledge out of them, building knowledge databases (such as ontologies [Gruber, 1993]) in an automated way.

The lowest level of grammar is morphology, therefore the automated learning of morphology represents the base of many grammar related problems. There are several approaches to the learning of morphology: some models apply classical methods like pattern matching, string transformation learning or classification, while others use some kind of artificial intelligence (AI) tools such as neural networks (NN) or genetic algorithms (GA).

In this dissertation I will examine the problem of the automated learning of morphological rules in order to generate and morphologically analyze inflected word forms. My approach is to solve these problems by applying classical methods. The proposed novel models are based on string transformation learning and pattern matching.

1.1 Problem domain

In this dissertation I explore the automated learning of morphology, the lowest level of linguistics that studies written language elements. Besides morphology, grammar has the following subdomains:

- Phonetics: the study of speech sounds
- Phonology: the study of the patterns of sounds in a language
- Syntax: the study of the structure of sentences
- Text linguistics: the study of texts
- Semantics: the linguistic study of meaning
- Pragmatics: the branch of linguistics that deals with the language in use and the contexts in which it is used

Morphology analyzes the internal structure of individual words and how the different word forms are constructed. 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. Higher level grammatical processing often uses the results of morphological analysis. For example during the syntactic analysis of a sentence we can use the morphological structure of its words to determine their syntactic roles. That is why morphology is the first step towards the automated processing of free texts.

DOI: 10.14750/ME.2022.020

According to morphology, words are constructed from **morpheme**s, that are the smallest grammatical units with associated meaning. As we will see in Section 1.2, some languages have stronger syntax and less morphological rules, thus less morphemes in their words, while other languages are morphologically more complex.

In concatenative morphology, words are built up from a lemma and a number of affixes. The **lemma** is the grammatically correct root form of the word that holds its base meaning, while **affix**es are added to the lemma to modify the base meaning. In natural languages there are a finite number of affix types that determine the semantic meaning of the affixes, i.e. how the meaning of the base form is altered by them. Examples of affix types include accusative case, plural form, past tense, etc.

Affixes can appear in arbitrary positions inside the word [Bauer, 2003]. Prefixes are prepended to the root form of the word (*in-correct*), while suffixes are appended (*fly-ing*). Infixes are substrings that are neither at the beginning, nor at the end of the word. They are very rare in most languages, an example is the Latin verb *vi-n-cō* where the *'n'* denotes present tense. In English most affixes are suffixes, but in Hungarian we can also find some affix types that use prefixes (e.g. *(megy, ki-megy)* which are *go* and *go out* in English) or both prefixes and suffixes (e.g. *(jó, leg-jo-bb)* which are the base and superlative form of the word *good*). This latter case, when an affix is made up of both a prefix and a suffix is often called a circumfix.

Another important morphological feature of a word is its **part of speech** (POS) that indicates its main syntactic feature. One word might have multiple possible parts of speech, and the part of speech of a word might change during inflection, when using derivative affix types. As an example, the word *good* is an adjective, but its inflected form *goodness* is a noun.

The process of adding affixes to a word is called **inflection**, while the inverse operation where we analyze the internal structure of a word is called **morphological analysis**. The input of inflection is a word and a set of affix types, while the output is the inflected form. The input of analysis is an arbitrary word, and the output is the lemma and the list of affix types found in the word. Morphological analyzers can sometimes also determine the intermediate word forms of the input word.

Morphological analysis represents a more complex problem than inflection, since we do not know how many and what kind of affixes to look for. Inflection rules can be as simple as prepending or appending some characters to a word, but in some languages they often cause other side effects such as vowel or consonant gradation. In such cases the base form of the word also changes, making both inflection and analysis more difficult to execute.

A simpler variation of morphological analysis is called **segmentation**. In this case, the output does not contain the affix types, only the morphs, i.e. the substrings identified by affix boundaries. **Stemming** is a simplification of segmentation, when only the base form is returned. The **stem** is the word form that we get if we drop all the affixes. The stem and the lemma are often identical, unless the base form changes. For example the lemma of the English word *tried* is *try*, and its stem is *tri*. On the other hand, both the stem and the lemma of the word *dogs* is *dog*. **Lemmatization** is another variation of morphological analysis, when the lemma of the given word is determined. Lemmatization is more complex than stemming, but simpler than a complete morphological analysis.

DOI: 10.14750/ME.2022.020

1.2 Natural Language Categorization

Some morphology models are optimized for a given set of natural languages by including language specific information and knowledge into the algorithms, while others use statistical methods and can be applied to a wide variety of languages.

Natural languages have different classification systems, based on for example their historical properties, word order and morphological attributes. While the latter one directly relates to morphology, the second one is more related to syntax than to morphology. The historical categorization is important mainly for language researchers.

1.2.1 Historical Language Families

Based on the history of languages, linguistics defined different language families [Akmajian et al., 2017] that can be displayed in a tree. The child nodes of the root are so-called proto-languages that usually represent the common root of the languages of a continent. Under the proto-languages, there are several subcategories, until we drill down to the level of leaves that contain the actual spoken languages themselves.

Figure 1.1 displays the relationship of the Indo-European languages. Other proto-languages are Uralic, Caucasian, American, Austroasiatic and Sino-Tibetan. The languages in the same family usually have similar morphological and syntactic attributes.

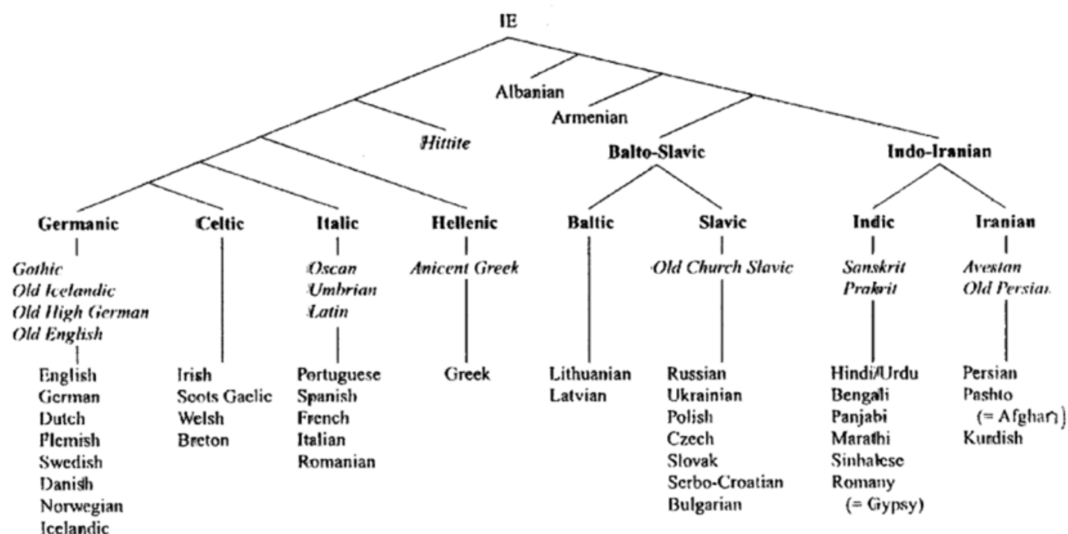


FIGURE 1.1: The Indo-European language family

As we can see in Figure 1.1, some European languages are missing, since they are not Indo-European languages. For example, the Hungarian and Finnish languages are Uralic languages, meaning that their roots originate from the Ural Mountains. Therefore these languages are somewhat different from Indo-European languages, but they have several similarities to each other.

1.2.2 Linguistic Categories

This classification system is closer to syntax than to morphology, because it describes word order inside the sentences. Usually if a language has fewer inflection rules,

DOI: 10.14750/ME.2022.020

then information is encoded using other techniques, such as word order or auxiliary words.

There are nine categories in this classification system like languages with Subject-Object-Verb (SOV) or Subject-Verb-Object (SVO) word order [Meyer, 2009]. An example sentence for SVO is *"He has eaten an apple."* in English.

In Hungarian we can use multiple word orders based on the main information that we want to communicate. The main word order is SVO: *"Ő evett egy almát."* (*"He ate an apple."*) but we can also move the words around to emphasize one part of the sentence: *"Ő egy almát evett."* where the fact that he ate an apple and not an orange for instance is more important than the fact that he ate something.

1.2.3 Morphological Language Classes

Based on their morphological aspects, languages can be grouped into several different classes [Gelbukh et al., 2004, Akmajian et al., 2017].

Analytic languages have a fix set of possible affixes for each part of speech, and usually each word has a low morpheme-per-word ratio. Instead of affixes, grammatical modifiers are encoded using word order and auxiliary words. English has some analytic aspects.

Isolating languages are similar in that their words have usually no affixes, each word is its own stem. Examples of such languages are Chinese or Vietnamese.

Synthetic languages, unlike the isolating and analytic languages have a high morpheme-per-word ratio, meaning that grammatical relations are expressed by adding different affixes to the lemma. This language category has three subgroups:

- **Polysynthetic languages** have very complicated grammatical rules. Some Native American languages belong to this class, where the semantic meaning of words is equivalent to the sentences of more modern languages. This means that each word consists of several morphemes. For example the Inukitut word *tavvakiqutiqarpiit* means *"Do you have any tobacco for sale?"*.
- Many Indo-European languages fall into the category of **fusional languages**. The name of this language class comes from the fact that their words usually fuse multiple affixes into just one, combining several grammatical relations. Therefore the morphemes are hard to be distinguished from each other, their boundaries are blurred. Vowel and consonant gradation, as well as suprasegmental features such as stress and tone are also frequently used to modify semantic meaning. Russian, Polish, Slovak and Czech languages fall into this category among others.
- The third subcategory contains the **agglutinative languages** like Hungarian, Finnish, Turkish, Japanese, Korean, Aztec or Esperanto. In these languages the words also contain multiple affixes, but their boundaries are easier to find than the boundaries of fusional affixes. The morphological complexity of this language class comes from the fact that each word can contain many affixes, and the base form is often transformed, too.

There are other exotic languages that do not follow the rules of concatenative morphology. **Intraflective languages** like Arabic and Hebrew express the meaning of the words using consonant characters, while vowels add the grammatical meaning to them. For example the Arabic word *kitab* has the lemma of *k-t-b* and the affixes of *-i-a*. In the Ngiti language [Booij, 2012] the plural form of a noun is generated by replacing the last two syllables with high tone syllables. For example the singular *kama* has the plural form *kámá*.

DOI: 10.14750/ME.2022.020

1.3 Research Goals

Due to the complexity of morphology and the high number of irregularities, it is a big challenge in computational linguistics to develop an efficient learning algorithm for induction of inflection rules. Chapter 2 will summarize the currently available models, but most of them have some drawbacks. The goal of this research is to develop an efficient pattern matching based novel morphology model that can

- learn inflection rules from a training data set,
- inflect lemmas using a set of affix types, and
- analyze given words, determining the morphological structure of the input, its lemma, possible parts of speech, affixes and its intermediate word forms.

Figure 1.2 displays a simplified view of the proposed model, including its main components.

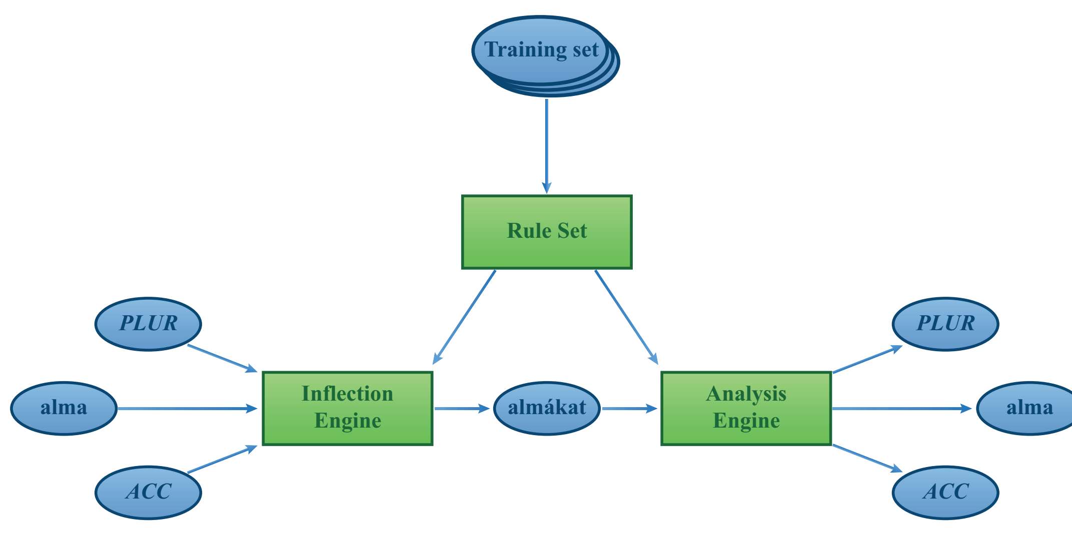


FIGURE 1.2: The main components of the proposed morphology model

From a training data set, the model can deduce a rule set that is used by the inflection and analysis engines. The inflection engine can output inflected word forms based on input lemmas and affix type sets, while the analysis engine outputs the lemma and the found affix types in the given inflected form. As we will see in later chapters, the output includes the intermediate word forms as well, and the given responses are sorted based on a calculated confidence value.

The target language of this research is Hungarian, which is a morphologically complex agglutinative language, containing many affix types, complex inflection rules including many vowel and consonant gradations. In Hungarian, a noun can have 1,400 related variants, and a verb can have 59 different forms according to [Prószéky and Novák, 2005]. The reason why this language was chosen for the evaluation of the proposed model besides its morphological complexity is that this is my native language. However, the proposed model is likely suitable for other languages as well, mainly those that are in the same or less complex language category as Hungarian. Nonetheless, the evaluation of the model will be performed using Hungarian data sets. Just to illustrate the diverse nature of Hungarian inflection rules, let us look at Table 1.1 that contains the base form and the accusative case (or cases) of some Hungarian words.

DOI: 10.14750/ME.2022.020

TABLE 1.1: The accusative case of some Hungarian words

Index	Lemma	Accusative case	Meaning
1	*papír*	*papírt*	*paper*
2	*bot*	*botot*	*stick*
3	*kefe*	*kefét*	*brush*
4	*malom*	*malmot*	*mill*
5	*tó*	*tót* or *tavat*	*lake*

The main rule in case of Hungarian accusative case is to append a *'t'* character at the end of the word (1). However, sometimes we also need to insert an extra vowel that depends on the vowels of the base form (2). If the word ends with a short vowel, it may become long (3). There are many exceptions as well that have more complex inflection rules (4), and some words may have multiple possible valid inflected forms as well (5). Almost every affix type has its own complexities and exceptions.

The novel scientific results will be evaluated using several test scenarios, and also compared with the available baseline model implementations from literature to demonstrate the advantages of the proposed model.

1.4 Dissertation Guide

The structure of the dissertation will be as follows:

- In Chapter 2 I present the survey of the main morphology models that can be found in literature. In this survey we can read about the main concepts of these models, their advantages and disadvantages.
- Chapter 3 compares and analyzes the main existing Hungarian morphological analyzers and their annotation token systems using objective metrics. These metrics, the analysis model and the evaluation results form the basis of my first thesis.
- One of the key components of an efficient morphology model is a transformation engine that can induce transformation rules from a training word pair set demonstrating the characteristics of a single affix type. In Chapter 4 we can read about two such novel models: the first one is a lattice based model, while the other one is a string based transformation engine called ASTRA (Atomic String Transformation Rule Assembler). The novel scientific results of these models form the basis of my second thesis.
- Chapter 5 contains the formalism and evaluation of the novel Morpher model. While ASTRA is a single-affix morphology model that can learn the inflection rules of a single affix type, Morpher is a multi-affix morphology model. It can learn not only the transformation rules of the affix types of the target language, but also the conditional probabilities of the valid affix type chains, the lemmas and their parts of speech. The novel scientific results of the Morpher model form the basis of my third thesis.
- In Chapter 6 I examine the space and time complexity of the Morpher and ASTRA models. After understanding the cost of the training phase, the inflection and the morphological analysis operations and their substeps, I introduce three possible optimization techniques that can be used to reduce the

DOI: 10.14750/ME.2022.020

rule base size (and thus the average inflection and analysis time) by eliminating unnecessary rules during the training phase. The novel scientific results of the cost analysis and the introduced optimization techniques form the basis of my fourth thesis.

- In Chapter 7 I describe the ways one can use the above mentioned novel morphology models for further research. The Java source code of these models, including the lattice based transformation engine, ASTRA and the Morpher model is available on Github. They are also easy to consume using Maven and Gradle through the jcenter or Maven Central repositories. I also developed a Spring Boot based server-side REST API application that publishes the main operations of Morpher, and published a Docker image of it so that research projects developed in other software environments can consume these operations as well. For users, a multi-platform React and React Native based client-side application is also available to use. These implementations form the basis of my fifth thesis.
- In Chapter 8 I summarize my main contribution to this scientific topic, including all the novel scientific results of this dissertation and the possible future research areas that can further improve the proposed models, or use them to solve higher level NLP problems.

DOI: 10.14750/ME.2022.020

Chapter 2

Survey of the Current Models

In this chapter I summarize the main aspects and features of existing morphology models found in literature. Besides introducing the most important approaches of morphological analysis and inflection, as well as automated learning methods, my goal is also to categorize these models based on different grouping methodologies.

In Section 2.1 I examine several dimensions of the morphology models found in literature. After that, in Section 2.2 I introduce the existing baseline models that I will compare with the proposed models in the following chapters. Section 2.3 summarizes the conclusions of the literature research, highlighting the advantages of the examined models and those areas that I want to improve with my proposed models.

2.1 Categorization of the Existing Morphology Models

During the literature research, I found several dimensions that we can use to categorize the examined morphology models. The dimensions that will be explored in the following subsections include:

- Knowledge representation: What kind of information is extracted from the training data and how? How well the examined model can generalize from the training data?
- Scope: Can the examined model handle all the affix types of the target language and recognize affix chains in the input words, or was it designed to learn the transformations of only one affix type?
- Symmetry: Can the examined model both inflect and analyze input words using the same knowledge base, or does the stored knowledge concentrate only on one direction?
- Granularity of analysis: In case of morphological analysis, what is the interface of the model? Can it return a complete response containing intermediate words and the lemma (analysis) or only the affix boundaries (segmentation)? Can it determine the grammatically correct root form (lemmatization) or does it only drop the affixes from the input word (stemming)?
- Machine learning capabilities: How much annotated data, a priori knowledge is required for the training phase? There are supervised, unsupervised and semi-supervised models according to this aspect. However, several models exist where the knowledge base has been built up manually by human experts.

2.1.1 Knowledge Representation

One of the most important questions regarding morphology models is their training method and learning algorithm. The key question is how these models represent knowledge in an easily searchable way. Most models that I found in literature use

DOI: 10.14750/ME.2022.020

either dictionaries, extract morphological rules or apply statistical learning methods, often even combining these techniques. Nowadays, artificial intelligence based methods are also used frequently to solve morphological problems.

Dictionary Based Systems

As Figure 2.1 demonstrates, dictionary based systems can be imagined as sets of key-value pairs, where we map from the left-hand side to the right-hand side. In morphology, the mapping happens from the base word forms to their inflected forms or vice versa.

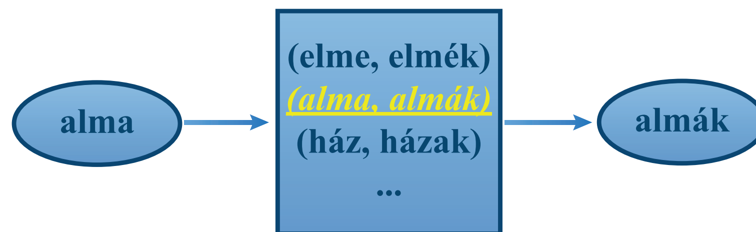


FIGURE 2.1: A simplified view of dictionary based systems

In Figure 2.1 we can see some of these key-value pairs: the base form and plural of the Hungarian words for *mind*, *apple* and *house*. When the dictionary receives the word *apple*, it looks up the appropriate key-value pair and returns the correct plural form.

As we can see, dictionaries can store the inflected forms of one affix type easily. However, the dictionary size can increase dramatically if we want to include all the possible inflected forms. Another disadvantage of dictionaries is that they cannot generalize well: if an entry is missing from the dictionary, then the model will not be able to produce its inflected form. Moreover, dictionaries are often built manually by human experts, as pointed out in [Gelbukh and Sidorov, 2003].

Therefore such simple dictionaries are usually only used to store the irregular word forms as part of more complex rule based or statistical morphology models, as described in [Tóth and Kovács, 2014]. The AMC model uses a smaller sized associative memory to store the exceptions that would otherwise be more difficult to handle correctly by the proposed classification model.

Another grammar related application area of dictionaries is WordNets that exist for both English [Miller, 1998] and Hungarian [Miháltz et al., 2008] among other languages. However, they tend to focus more on the syntactic and semantic levels than morphology. Also, due to the complex nature of grammar, WordNets leverage ontologies for knowledge representation. Theoretically they could contain morphological information as well, but it would increase their size and connections radically. The Szeged Corpus [Csendes et al., 2004] is one such attempt that includes 1.2 million word entries that have been morpho-syntactically analyzed using Humor [Prószéky and Tihanyi, 1993] and then manually disambiguated.

The content of dictionaries are often compressed using different data structures. One possibility is to use finite state transducers (FSTs) [De la Higuera, 2010] that will be further explored in Subsection 2.2.2.

DOI: 10.14750/ME.2022.020

Rule Based Systems

Instead of storing the exact words and word pairs of the training data, rule based systems try to extract compact transformation rules from it and store these rules instead, as shown in Figure 2.2.

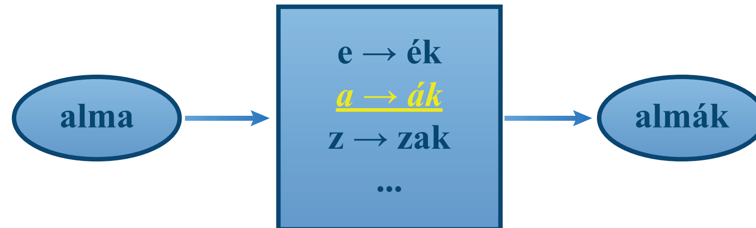


FIGURE 2.2: A simplified view of rule based systems

This conceptual difference results in a smaller knowledge base, since one rule can cover several items in the training data set. This also helps in generalization, since the stored transformation rules might match words that have not been included in the training data set, but can still be transformed correctly using the rule base. However, sometimes the matching transformation rule results in an incorrectly transformed word form, this case is called overgeneralization. The overgeneralization effect can be decreased by storing a smaller number of exceptional cases in a dictionary for instance [Tóth and Kovács, 2014].

The main parts of a transformation rule is the context and the replacement. Models differ in how they represent the substring to be changed and the replacement string, as well as how much information is stored about the context. There are simpler context insensitive methods that only care about the substring that needs to be replaced in the input word, while context sensitive methods can also distinguish among the different occurrences of the same substring based on its index or neighboring characters.

One of the best-known rule based systems is the Porter stemmer [Porter, 1980] that can cut the affixes of the input words using simple transformation rules. The model was originally designed for the English language and contained 60 base rules. The stemmer was then generalized as a simple programming language called Snowball [Porter, 2001] that can be used to define the rules. A compiler can create a thread-safe C or Java based application from the Snowball source code. For the Hungarian language, several Snowball based stemmers were created, but due to the complex agglutinative nature of the language, it turned out that they all performed worse than the best n-gram based solution [Tordai and de Rijke, 2006].

A more complex rule based model was established by Hajič for the Czech fusional language [Hajič, 1988]. The proposed model is based on a controlled rewriting system (CRS). On top of this CRS model, a substitution operation is defined that can replace parts (variables) in words, using rewrite rules. The same variable is replaced with the same string in all occurrences. Although the formalism of the controlled rewriting system is a suitable morphology framework for agglutinative and fusional languages, the paper does not determine the rule generation process.

The TASR model [Shalonova and Flach, 2007] that will be examined in Subsection 2.2.3 is also a rule based system. The generated suffix rules are stored in a tree for efficient searching.

DOI: 10.14750/ME.2022.020

Statistical Methods

Statistical methods use some kind of probabilistic approach to generate the best knowledge base from the given training data set. The internal representation of these models can vary, they might store transformation rules as well, but the point is that they try to optimize the stored entities based on a goodness value calculated from the training data using some kind of probabilistic method.

One common approach is to define an error function either locally for the rules or globally for the knowledge base. In [Satta and Henderson, 1997] the generated rules are simple string transformation rules that have positive and negative evidence, i.e. the number of items in the training set for which these rules apply or not. The learning algorithm tries to find the set of generated transformations for which the global error value is optimal. For performance reasons, suffix trees are used extensively to store the substrings.

Markov models also form a popular statistical method family that has several variations. Pair hidden Markov models (PHMMs) have been tested to learn inflection rules successfully [Clark, 2001]. The goal of a PHMM is to learn string transductions from training word pairs based solely on probabilities. PHMMs consist of states, similarly to FSTs, where s_0 is the start state and s_1 is the end state. At each state, the model can output characters into the left stream (q_{10}), the right stream (q_{01}) or both (q_{11}). In each state, the sum of output parameters must be 1 for the alphabet Σ:

$$\sum_{c \in \Sigma} q_{11}(c \mid s) + q_{10}(c \mid s) + q_{01}(c \mid s) = 1$$

[Creutz and Lagus, 2004] utilizes hidden Markov models (HMMs) using a four-step unsupervised inflection learning algorithm. The model assumes that the words consist of a stem, and zero or more prefixes and suffixes. The initial segmentation is done using Morfessor Baseline [Creutz and Lagus, 2002], then it is adjusted using a first-order Markov chain (bigrams). The evaluation using Finnish and English corpora shows that although the Morfessor Baseline method improves its precision using larger training data, its recall drops, while the proposed model reaches lower precision, but its recall remains high.

In the work of [Lafferty et al., 2001], conditional random fields (CRFs) are proved to improve HMMs and maximum entropy Markov models (MEMMs). According to the publication that originally proposed the CRF model, CRFs are more robust than Markov models and do not have the label bias problem, even with the same parameterization. CRFs have been applied successfully for the Uyghur language that is a morphologically complex agglutinative language [Aisha and Sun, 2009]. For evaluation, more than 500 thousand words have been collected from the Internet, from various domains, and the proposed model reached about 90% accuracy.

A subcategory of statistical morphology models is those that treat inflection as a classification problem. Generally the input of classification is a set of $O = \{o_i\}$ objects and their classes in the form (o_i, c_j) where the ith object belongs to the jth class, $c_j \in C$. The goal is to extrapolate and learn the class of previously unseen objects as well. If the set of $O \subseteq U$ is the training set and $O' \subseteq U$ is the remaining set of objects for which $O \cap O' = \emptyset$ and $O \cup O' = U$, then we need to define the $m : U \to C$ membership function, knowing only its projection $m' : O \to C$.

In case of morphology, the training set will contain word pairs $(w_{il}, w_{ir}) = o \in O$ containing the base form and inflected form of the word pairs according to an affix type. The classes will be the transformations that need to be applied on the base forms. During inflection, a classification model needs to find the closest known word

DOI: 10.14750/ME.2022.020

to the input word and its class, then apply the transformation represented by the class. The AMC model [Tóth and Kovács, 2014] uses a classification engine at its core to find the inflected form of input words.

Artificial Intelligence Based Methods

As with any other problem domain, artificial intelligence (AI) models can also be used. The main prerequisite of solving the inflection problem with an AI model is to translate the inputs and outputs (in this case, words and transformations) to their domain, often numeric values.

Neural networks (NNs) are relatively easy to use, since they have firm mathematical background and several implementations can be used out of the box. In most NN based morphological solutions, words are treated independently, meaning that related word forms have no stored connections. [Luong et al., 2013] combines recursive neural networks (RNNs) that can manage the morphemes of words, and neural language models (NLMs) that manage words in sentences or phrases. To segment the words for evaluation, the Morfessor [Creutz and Lagus, 2002] model was used, that will be examined in Subsection 2.2.4. Most SIGMORPHON models use an NN variant as well, these models will be introduced in Subsection 2.2.5.

Another popular AI model that can be treated as a black box is the model of genetic algorithms (GA). [Gelbukh et al., 2004] proposes a morphological application, where the base forms and endings of the training words are extracted into a knowledge base in an unsupervised manner. The main concept behind genetic algorithms is that some key features of the problem are identified as genes that are modified using crossover and mutation in some iterations. This model is not deterministic, but choosing the right genes and parameters, as well as the right number of iterations usually leads to a valid solution. The proposed model was evaluated using the words of the Scrabble game, as well as the Spanish words of Don Quijote. For English words, the Porter stemmer had better accuracy.

2.1.2 Scope

Morphology models can also be categorized based on their scope, i.e. if they are capable of learning the whole morphology of a language including all of its affix types, or only the transformations of a single affix type.

Single-Affix Models

Some of the already mentioned models can only handle a single affix type. For instance a simple dictionary stores word pairs and cannot distinguish among the transformations of different affix types. Also, FSTs [De la Higuera, 2010] and the TASR model [Shalonova and Flach, 2007] receives a word pair set as their training data, without any information on affix types. They will be examined in Subsection 2.2.2 and Subsection 2.2.3, respectively.

The only way such tools can be used for multi-affix morphology learning is to develop a higher-level model for affix type chain management, and training a separate FST or TASR for each affix type, or extend their structure with some labeling mechanism.

DOI: 10.14750/ME.2022.020

Multi-Affix Models

Multi-affix models are capable of handling multiple affix types, i.e. inflect words using multiple affix types and recognize transformations related to multiple grammatical categories.

The Porter stemmer [Porter, 1980], although it does not know exactly the affix type labels, can cut multiple affixes from the input words.

Similarly, Morfessor [Creutz and Lagus, 2002], MORSEL [Lignos, 2010] and MorphoChain [Narasimhan et al., 2015] can find the affix boundaries of multiple affix types. However, they cannot label these affixes using affix type names, either. These models will be further explored in Subsection 2.2.4.

The SIGMORHPON shared tasks also require to model all affix types of the target languages, so the models of Subsection 2.2.5 also fall into the multi-affix model category, as well as the morphological analyzers of Subsection 2.2.6.

2.1.3 Symmetry

Another simple dimension of morphology model categorization is symmetry: some models are unidirectional, meaning that they can either inflect or analyze the input words; others can work in both directions.

Asymmetric Models

The main constraint on symmetry is the structure of stored knowledge. For instance a classic dictionary based system usually can only work in one direction. Therefore if the keys are the base forms, then such a dictionary can only execute inflection efficiently, and not analysis.

FSTs [De la Higuera, 2010] that will be explored in Subsection 2.2.2 are also asymmetric, since the states are built up in such a way that the input words are treated as the base form and the output channel will contain the transformed inflected form.

The TASR model [Shalonova and Flach, 2007] that will be presented in details in Subsection 2.2.3 has a tree representation of suffix rules that is built up using the characters of the base form. Therefore such a tree cannot handle backwards transformation, only forwards transformation.

There are some models that were proposed for solving the opposite problem. Linguistica [Lee and Goldsmith, 2016] (explored in Subsection 2.1.5), as well as the Morfessor model [Creutz and Lagus, 2002], the MORSEL model [Lignos, 2010] and the MorphoChain model [Narasimhan et al., 2015] (explored in Subsection 2.2.4) can be used for segmentation.

Hunmorph-Ocamorph [Trón et al., 2005], Hunmorph-Foma [Hulden, 2009], Humor [Prószéky and Tihanyi, 1993] and Hunspell [Pirinen et al., 2010] are morphological analyzers (more details in Subsection 2.2.6), while the already introduced Porter stemmer [Porter, 1980] can only be used for stemming purposes.

Some of these models including FSTs and TASR can be extended to be used for both forwards and backwards transformation of a single affix type. If the original training word pair set is reversed, then the trained model will be able to handle backwards transformation. This means that if we use two transducers or two trees, one built using the original training data and the other one built using the reversed training data,[1] then theoretically we will be able to handle both directions.

[1]We can always generate the reversed training data if we swap the base and inflected forms in the training data set. This means that instead of *(apple, apples)*, we use *(apples, apple)* to train the the affix type of plural form.

DOI: 10.14750/ME.2022.020

Symmetric Models

Symmetric models are very rare in literature, probably because usually the new models are proposed to solve a unidirectional morphological problem.

Some of these models (including FST or TASR) are easy to extend for the bidirectional case. In case of a simple dictionary based system, it is also possible to achieve the same using a bidirectional mapping.

2.1.4 Granularity of Analysis

Analysis models can be also categorized based on how complete their response is. Some models return the intermediate words and the lemma with the affix types and part of speech, while other models can only segment the input words and cannot handle changing base forms. Regarding the base form, lemmatization models can determine the grammatically correct root form, while stemmers can only drop the affix characters.

Morphological Analysis

Lemming [Müller et al., 2015] is the first log-linear model that handles both morphological analysis and lemmatization. It works on the token level and is able to lemmatize unknown word forms, without the need of providing morphological dictionaries or external analyzers.

The morphological analyzers of Subsection 2.2.6 also return a relatively granular response, containing the grammatically correct lemma, its part of speech and the affix types found in the input word.

Segmentation

Linguistica [Lee and Goldsmith, 2016], that will be introduced in Subsection 2.1.5, as well as Morfessor [Creutz and Lagus, 2002], MORSEL [Lignos, 2010] and MorphoChain [Narasimhan et al., 2015] (introduced in Subsection 2.2.4) are a few popular unsupervised segmentation models. Their common feature is that they concentrate on affix boundaries and use statistical methods to determine the highest probability segmentation of the given word set. However, they are unaware of the affix types of the language, and moreover they cannot handle base form transformations like vowel or consonant gradations.

Lemmatization

Lemmatization models can return the grammatically correct root form of the given word even if its base form was modified by the applied affix types.

The previously mentioned single-affix models are capable of detecting base form transformations, including dictionaries, FSTs [De la Higuera, 2010] and the TASR model [Shalonova and Flach, 2007]. The FST model will be explored in details in Subsection 2.2.2, while Subsection 2.2.3 will introduce the TASR model.

Hunmorph-Ocamorph [Trón et al., 2005], Hunmorph-Foma [Hulden, 2009], Humor [Prószéky and Tihanyi, 1993] and Hunspell [Pirinen et al., 2010] can also be mentioned as lemmatization tools, because if they can analyze the given word, they also return its lemma. They will be detailed in Subsection 2.2.6.

DOI: 10.14750/ME.2022.020

Stemming

Stemmers differ from lemmatizers in that they cannot recognize base form transformations, so they can only identify the affix substrings and drop their characters from the word. The Porter stemmer [Porter, 1980] is one of the most famous examples, also adapted for the Hungarian language [Tordai and de Rijke, 2006].

2.1.5 Machine Learning Capabilities

Nowadays the emphasis is transposed from supervised models towards unsupervised and semi-supervised models that require less and less input knowledge and preprocessed data. However, there also exist models whose knowledge base is built up by human experts in a non-automated way.

Non-Automated Training

In case of non-automated training, the model is built by human experts, thus the model construction method includes no automated steps.

Historically, the first models were built like this. Dictionaries and even the rules of the Porter stemmer [Porter, 1980] are part of this category.

However, later supervised methods emerged that could build the knowledge base from preprocessed data in an automated way.

Supervised Training

Supervised models require a training data set consisting of records that match a fixed set of constraints. The big disadvantage of supervised models is that this training data needs to be created somehow, usually in a non-automated way.

The FST [De la Higuera, 2010] and TASR [Shalonova and Flach, 2007] models for instance (as we can read in Subsection 2.2.2 and Subsection 2.2.3) require a word pair set that demonstrate the transformations of a single affix type of the target language. From such a word pair set they can build the appropriate transducer and tree, respectively, that can later be used during either inflection generation or morphological analysis.

[Ahlberg et al., 2015] leverages complete inflection tables of the target language, and generalizes them into more compact paradigms, i.e. rule patterns. During inflection, the original inflection tables are reconstructed for the previously unseen input base words. The pattern matching algorithm is based on the longest common subsequence method, and a support vector machine (SVM) is used for classification during the inflection table reconstruction. This model was tested with several languages and performed relatively well. However, for its training, complete inflection tables had to be constructed manually. A similar model is presented in [Hulden, 2014], that leverages the Foma [Hulden, 2009] framework to find the longest common substrings.

[Cotterell et al., 2015] uses a special version of CRF, the semi-Markov CRF. The advantage of the proposed model called Chipmunk is that it can not only segment words, but it also has a wide variety of labels and explicitly models morphotactics. This means that unlike most other segmentation tools, this one uses labeled morphological segmentation, providing more information in its output. The model provides both the lemma and the stem of the input words, as well as their affix types and their related substrings.

DOI: 10.14750/ME.2022.020

Unsupervised Training

Linguistica was one of the major unsupervised segmentation models originally developed around the millenium [Goldsmith, 2001]. Its functionality grew gradually over the years: at first, stemming was the main problem it wanted to solve, then the identification of suffixes and prefixes. The cutting of suffixes happens in a recursive fashion: first, the last suffix candidates are cut, then the remaining stems are processed again. From the training word set, after the cutting phase, so-called signatures are generated that describe the possible transformations of the words falling into the category of the signature. For example, *NULL.ed.ing.s* means that words can be extended with the *-ed*, *-ing* and *-s* suffixes. The main concept of the learning method behind Linguistica is the minimum description length (MDL), where the goal is to find the most compact description of the trained model and the data it describes using information theory principles. The trained model itself contains three major components: the list of stems, the list of affixes and the list of signatures that determine which stems can appear together with which affixes.

The original publication mentioned several future plans, including the identification of related stems (allomorphs) and compounds, as well as the grouping of related signatures into so-called paradigms for performance reasons. Some of these plans have been implemented in later versions [Goldsmith, 2006]. However, the main deficiency of the model itself is that it was not tested using languages with complex morphology such as Hungarian or Finnish, as the goal was to develop a model for the more widely studied European languages. The evaluation included English, French, German, Spanish, Italian, Dutch, Latin and Russian. The results show that the accuracy was around 70%, but if the incorrect results of words that could not be reconstructed from the training data are omitted, the accuracy increases to 80%.

The most recent version of Linguistica at the time of writing is Linguistica 5 [Lee and Goldsmith, 2016] that is a complete rewrite of the older C/C++ based engine in Python. The main benefit of the latest version besides the several improvements in its model is that it can function both as a graphical user interface that is easy to use and as a library, serving higher-level client applications.

The Paramorph model [Snover and Brent, 2003] is designed for concatenative morphology. It assumes that words consist of stems and suffixes, but contains no morphological ambiguity handling, does not support multiple interpretations of the same word and does not distinguish between derivational and inflectional affixes. The learning algorithm is based on statistical computation and has no knowledge about the target language. The main components of the model are a generative probability model and a two-phase search algorithm, consisting of a hill-climbing and a direct search phase. The evaluation was done for the English and Polish languages, comparing the results with Linguistica [Goldsmith, 2001]. However, determining the accuracy of a segmentation ignores the affix boundaries, and only checks the generated stems. Also, Linguistica proved to be significantly faster, and it reached higher accuracy in case of larger word sets.

Since the birth of Linguistica, there have been several other models that were built upon it. One of them is [Goldwater and Johnson, 2004] that uses the output segmentation of Linguistica and adds new components to the resulting model. The major added value is the introduction of phonological rules that consist of a transformation, e.g. $\epsilon \rightarrow e$ or $y \rightarrow ied$, and a context. The context is a simple string containing four characters, $X_t y_t y_f X_f$ where X_i holds the information if the character is a vowel, a consonant or a word-end symbol, and y_i is an arbitrary concrete character. The t index denotes the stem, while the f index denotes the suffix. These

DOI: 10.14750/ME.2022.020

rules are generated from the segmentation output of Linguistica in an unsupervised manner.

[Soricut and Och, 2015] presents a rather original segmentation model compared to the other MDL based ones. It treats words as n-dimensional vectors in vector space and tries to find the transformations among them. The resulting trained model can be visualized as a graph, where the nodes are the words and the edges are the transformations among the words. The transformation rules are in the form of *type:from:to*. For example, the rule *suffix:ϵ:ed* describes the past tense of many English words. Evaluation was performed on various languages including English, German, French, Spanish, Romanian, Arabic and Uzbek, and accuracy reached around 80-90%.

Semi-Supervised Training

Minimally supervised models are getting more and more attention lately, due to the fact that they only require a smaller set of annotated training data and a larger unannotated data set [Ruokolainen et al., 2016].

[Tepper and Xia, 2010] proposes a model that extends and tries to improve the Morfessor Categories-MAP model by incorporating a small amount of manually created morphological rules. The resulting method can not only provide the segmentation of the input words, but also their surface-level components. After the preprocessing takes place using Morfessor, a word-resegmantion stage is executed that fine-tunes the original segmentation. The two main steps can be executed multiple times after each other incrementally. The proposed model was tested using the English and Turkish languages, but only reached 75-80% of F-score at most.

In [Ruokolainen et al., 2014], a new CRF-based semi-supervised model is proposed that differs from its predecessors in that it does not rely on only annotated data. The advantage of the training algorithm is that it can consume a large number of unannotated data and also take into account a smaller number of gold standard data. During segmentation, the characters of the input words are categorized as the beginning of a multicharacter morph (B), middle of a multi-character morph (M) and the single character morph (S) categories using a linear-chain CRF model distribution. The unannotated data is processed using the Morfessor model and then its output is converted and incorporated into the knowledge base of the proposed model. Evaluation using the MorphoChallenge 2009/2010 data set showed that the F-score reached about 80-90% for the English, Finnish and Turkish languages.

In [Faruqui et al., 2015], a model of inflection generation is presented as a sequence to sequence transducer. The model uses a recurrent neural network model and combines supervised learning with the processing of a larger unannotated data. Originally such models were used for machine translation of sentences, but this proposed model adapts the NN based learning algorithm for learning inflection rules. The experiments show that the model can reach more than 90% of accuracy for data sets containing German, Finnish, Spanish, Dutch, French and Czech examples.

In [Cotterell et al., 2015], a labeled semi-supervised morphological segmentation (LMS) engine is presented, that explicitly models morphotactics. The engine, that uses a rich label set, can be used for morphological segmentation, stemming and morphological tag classification. A probabilistic CRF model is used internally to determine the corresponding labels in the input words.

[Müller and Schütze, 2015] compare four variants of semi-supervised morphological tagging models: word-feature cooccurrence matrix with singular value decomposition, statistical language model for word clusters, CW embedding based on

DOI: 10.14750/ME.2022.020

neural network and accumulated tag counts format. The paper includes six different languages for evaluation. As a conclusion, the statistical language model outperformed the others in all tasks and for all languages. Another summarizing comparison study was published in [Ruokolainen et al., 2016]. This paper involved three main methods: the Morfessor method family, the adaptor grammar framework and the CRF approach. Similarly to the previously mentioned comparison, the statistical, CRF-based method proved to be the most efficient here as well.

A sequence to sequence transducer based inflection generation model was published in [Faruqui et al., 2015], that transforms its input characters to a sequence of output characters, representing the inflected form. The training set contains word pairs of the lemmas and inflected forms, but unlabeled data is also added to the training set. The experiments show that the model achieves better or comparable results to state of the art methods.

Another interesting problem in morphology is the so-called morphological reinflection problem, that means that the input word is an inflected form, and the goal is to generate another inflected form. This problem has been solved using a character based encoder-decoder recurrent neural network [Kann and Schütze, 2017] among others. The proposed method uses a probabilistic log-likelihood model. According to the presented experiments, the loss of performance for reducing the training data varies a lot between languages and using 4 times more unlabeled examples mostly obtains the highest accuracy.

Semi-supervised training models are similar to active learning, which is a special case of machine learning, where during the training phase, the training algorithm will interactively ask the user to provide annotations for some selected training records. The main difference between the above introduced semi-supervised models and the active learning method is that the above models are completely offline, meaning that no interaction is necessary. The advantage of this aspect is that human interaction does not slow down the training phase.

2.2 Main Baseline Morphology Models

In this section I describe the main baseline models that I will use in the comparisons of the subsequent chapters. Note that two-level morphology is only included for historical reasons and it is the only model that will not be used later.

2.2.1 Two-Level Morphology

One of the first general morphology models for morphologically complex languages was the two-level morphology model [Koskenniemi, 1983], that resides somewhere in between morphology and phonology. Since this is historically the first complete framework, I introduce its base concepts in this section, despite the models proposed in this dissertation follow different routes of morphology representation.

The main idea behind two-level morphology from which it got its name, is that words are represented on two related levels: the surface level contains the written form of the words, while the lexical level contains the morphological structure. The model uses a dictionary of valid lemmas and morpheme categories, as well as finite state transducers (FSTs) as the transformation engine, often combined into just one transducer. While the dictionaries contain language specific knowledge, the FST is language independent, and thus two-level morphology can be adapted easily for

DOI: 10.14750/ME.2022.020

several languages. The proposed model is usable for both directions, i.e. both for inflection generation and morphological analysis.

The main building blocks of the model are two-level rules that have several possible forms. The basic structure of a two-level rule is: $CP\ OP\ LC - RC$, where

- CP is the correspondence part, i.e. a concrete or abstract character pair of the two levels.
- OP is the operator that indicates the relation of CP and the context.
- LC is the left context, while RC is the right context.

Elementary rules have the following subtypes:

- Context restriction rules ($CP \Rightarrow LC - RC$) define an environment where a given correspondence can occur. This form means that CP can only occur if it is enclosed by the given context.
- Surface coercion rules ($CP \Leftarrow LC - RC$) define that for any pairs that have the same lexical character as CP, the surface character must be the same as in CP within the given context.

Elementary rules can also be combined into more complex composite rules. For example the rule $CP \Leftrightarrow LC - RC$ combines the above two elementary rules.

At the time of the first publication, the FSTs had to be created manually, which was a non-trivial task to do. Later, different two-level rule compilers were created that could generate the FSTs from two-level rules, but these rules still had to be created by human experts.

Theron and Cloete [Theron and Cloete, 1997] proposed an easier method to generate the rule set from word pairs based on edit-distance similarities of the base and inflected forms. The algorithm learns the two-level transformation rules, calculating the string edit difference between each source-target pair and determining the edit sequences as a minimal acyclic finite state automaton. The goal is to have two-level rules whose context is long enough to uniquely identify the transformation position, but not too long to be overspecified. To acquire optimal two-level rules, a directed acyclic graph is used.

One of the main issues related to the two-level morphology model is the computation complexity of the implementations. It was shown that it is inefficient to work with complex morphological constraints [Barton, 1986], if there are complex dependencies among the different morphemes.

2.2.2 Finite State Transducer (FST)

Finite state transducers (FSTs) can be built from a set of word pairs, and are often used to learn string transformations [De la Higuera, 2010].

The theory of FSTs are related to the theory of finite state automata (FSAs). An FSA is a $\langle \Sigma, Q, q_\epsilon, \delta, F \rangle$ structure, where

- Σ is the alphabet,
- Q is a finite set of states,
- $q_\epsilon \in Q$ is the start state,
- $\delta : Q \times \Sigma \to Q$ is the state transition function and
- $F \subseteq Q$ is the set of final states.

A sample FSA can be seen in Figure 2.3. The automaton has 3 states and its alphabet contains 2 characters: $Q = \{q_\epsilon, q_1, q_F\}$ and $\Sigma = \{a, b\}$. The input string *aabb* is recognized by the automaton, but *baaba* is rejected, as the automaton stops in a non-final state q_ϵ.

String transformations can be modeled as transductions. A transduction is a $t \subseteq \Sigma^* \times \Gamma^*$ relation, $t = \{(s_1, s_2) \mid s_1 \in \Sigma^*, s_2 \in \Gamma^*\}$ where s_1 is the input word and

DOI: 10.14750/ME.2022.020

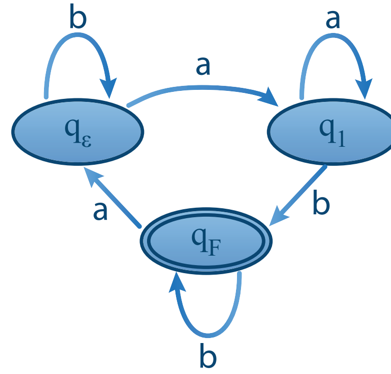


FIGURE 2.3: A sample finite state automaton

s_2 is the output word, while Σ is the input alphabet and Γ is the output alphabet. Such transductions can be learnt by transducers, that have several subtypes.

A rational transducer is a $\langle Q, \Sigma, \Gamma, q_\epsilon, E \rangle$ structure, where

- Q is the finite set of states,
- Σ and Γ are the input and output alphabets, respectively,
- $q_\epsilon \in Q$ is the unique start state and
- $E \subseteq Q \times \Sigma^* \times \Gamma^* \times Q$ is the finite set of transitions.

The addition on top of an automaton is the two alphabets and the transition function E that outputs a string besides changing the state.

A sequential transducer constraints E further so that it is $E \subset Q \times \Sigma \times \Gamma^* \times Q$ and $\forall (q, a, u, q'), (q, a, v, q'') \in E \Rightarrow u = v \wedge q' = q''$. This basically means that the transducer becomes deterministic.

A subsequential transducer is a $\langle Q, \Sigma, \Gamma, q_\epsilon, E, \sigma \rangle$ structure such that the first 5 components make up a sequential transducer and $\sigma : Q \to \Gamma^*$ is a total function, which adds outputs to the ending states of the processing. Internal states do not output strings.

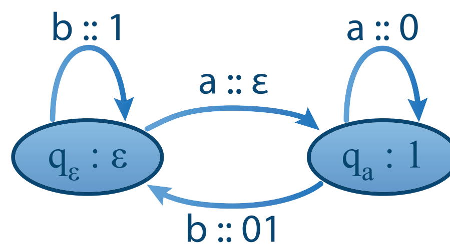


FIGURE 2.4: A sample onward subsequential transducer

A transducer is onward if $\forall q \in Q, a \in \Sigma$: lcp $(\{u \mid (q, a, u, q') \in E\} \cup \{\sigma(q)\}) = \epsilon$ where lcp returns the longest common prefix of the given word set and ϵ is the empty string.

Figure 2.4 displays a simple onward subsequential transducer with two states $Q = \{q_\epsilon, q_a\}$, using the input alphabet $\Sigma = \{a, b\}$ and the output alphabet $\Gamma = \{0, 1\}$. For the input string *aabba*, the FST will output *00111* and change its state from q_ϵ to $q_a, q_a, q_\epsilon, q_\epsilon, q_a$.

Originally, FSTs had been created manually by human experts, but nowadays automated training methods can be used as well. One way of building an FST from

DOI: 10.14750/ME.2022.020

a word pair set is the onward subsequential transducer inference algorithm (OSTIA) [Oncina et al., 1993, Oncina, 1998]. The training algorithm starts with building a prefix tree transducer containing states for every input word prefix. Every edge outputs an empty string, and every state outputs an output word form or a special character that does not exist in the output alphabet. Then, every character of the output is moved as close to the start state as possible, creating an equivalent onward prefix-tree transducer. Finally, since this transducer might contain hundreds or thousands of states, a complex recursive algorithm is used to merge certain states to get a minimal transducer.

One disadvantage of FSTs is that they cannot really generalize, they only act as compact dictionaries, returning the correct inflected form for known base forms [Mohri, 1997].

2.2.3 Tree of Aligned Suffix Rules (TASR)

The tree of aligned suffix rules (TASR) [Shalonova and Flach, 2007] is a very efficient model for learning the transformation rules from a training word pair set. The basic building blocks of TASR are the suffix rules.

A suffix rule is an $LS \rightarrow RS$ transformation where LS is the left-hand suffix and RS is the right-hand suffix. The $LS \rightarrow RS$ suffix rule is aligned with the $LS' \rightarrow RS'$ suffix rule if two words (w_1, w_2) and two prefixes (s, s') exist such that $w_1 = s + LS = s' + LS'$ and $w_2 = s + RS = s' + RS'$. The frequency of a suffix rule is the number of word pairs in the training set that the rule matches. The child suffix rule $LS \rightarrow RS$ is subsumed by the parent suffix rule $LS' \rightarrow RS'$ if there exists a character c such that $LS = c + LS'$ and $RS = c + RS'$.

The goal of the training algorithm of TASR is to extract the suffix rules from the training word pair set and build a tree from the extracted rules using the following constraints:

- In the root, $|LS|$ must be minimal, thus it is usually the empty string ϵ.
- The winning rule of a node has the highest frequency and is not subsumed by its parent rule.
- For every child node, $LS' = c + LS$ where $c \in \Sigma$ is a character and LS is the left-hand suffix of the parent rule.

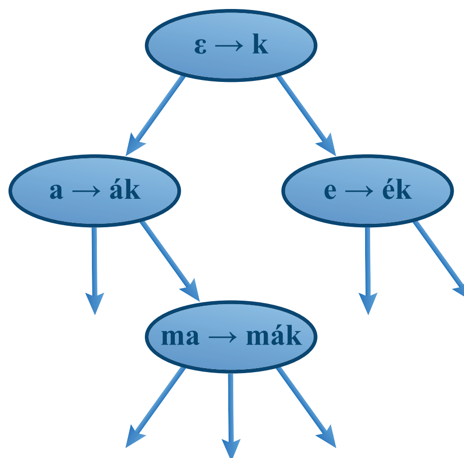


FIGURE 2.5: A sample tree of aligned suffix rules

DOI: 10.14750/ME.2022.020

This tree building algorithm can be executed quickly and easily, as described in one of my previous publications [9]. The excerpt of a resulting TASR tree can be seen in Figure 2.5.

During inflection generation, we must search for the closest matching pattern in the tree in a bottom-up fashion, and if we find a match, we must execute the transformation. For the input word *alma* (*apple*), we find the rule *ma* → *mák* and the output will be the correct plural form *almák*. For words of the training set, the exact word pair will be found in a leaf, while for previously unseen words, a generalized rule will be found with the longest matching substring.

TASR is a quick and efficient transformation engine that can learn suffix rules easily and even generalize the learnt rules. However, its disadvantage is that for affix types that contain prefix and circumfix rules as well (such as Hungarian preverbs or superlative), it cannot be used.

2.2.4 Unsupervised Segmentation Models

Morfessor [Creutz and Lagus, 2005a] is a segmentation model originally published in 2005. Since then, many publications have emerged that improved or extended it, such as MORSEL [Lignos et al., 2009] or MorphoChain [Narasimhan et al., 2015] that will be introduced in the following subsections.

Morfessor 2.0

Morfessor is an unsupervised and semi-supervised word segmentation method family consisting of concrete models like Morfessor Baseline, Morfessor Baseline-Freq-Length, Morfessor CategoriesML and Morfessor Categories-MAP.

The first public version of Morfessor Baseline [Creutz and Lagus, 2005b] was implemented in Perl and evaluated against Linguistica 5 [Lee and Goldsmith, 2016]. The main difference between these two models is that while Linguistica assumes that a word consists of a stem and at most one suffix (or prefix) and uses recursion for generalization, Morfessor has no upper limit on the number of prefixes and suffixes.

The main building blocks of words according to the Morfessor Baseline model are atoms (characters), constructions (morphemes) and compounds (words). The main goal of the training algorithm is to find the constructions (morphemes) from a given training word set. The process of replacing compounds with constructions is called tokenization.

Originally two cost functions were introduced [Creutz and Lagus, 2002]: the minimum description length (MDL) based learning and the maximum likelihood based learning. In case of MDL, the training process uses recursive segmentation, incrementally trying to add new morphs to the lexicon. As this method might lead to a local optimum, the model includes a so-called dreaming phase where the already processed words are processed again in random order. The maximum likelihood based learning is not recursive but linear, and starts with an initial random segmentation. If the model encounters rare morphs or a morph chain consisting of one-character morphs, it rejects them. The evaluation showed that for both Finnish and English words, the MDL based learning method was the best, producing the highest accuracy and smallest morph lexicon. Linguistica 5 generated a larger morph lexicon and was significantly worse for Finnish data than Morfessor. However, for English it produced better results, due to the simpler morphology of the language.

DOI: 10.14750/ME.2022.020

There have been several model variants such as Morfessor Categories-MAP or Morfessor Categories-ML [Creutz and Lagus, 2005a] that tried to alter the learning and search processes for higher F-score in case of highly agglutinative languages such as Finnish.

In 2013 the original Morfessor Baseline model has been reworked and improved [Virpioja et al., 2013]. The new Morfessor 2.0[2] was implemented in Python, using an easy to use command-line and library interface. Besides the more modern architecture, Morfessor 2.0 improves the original Morfessor 1.0 Baseline by supporting training speed-up using random skips, semi-supervised training, online training, etc.

The semi-supervised training aspect of Morfessor 2.0 means that the model can receive a smaller sized annotated and a larger sized unannotated data, and it can tune the internal parameters of the learning algorithm to achieve the best possible results using the two data sets. This means that the training can happen without annotated data (unsupervised mode), but adding just a few annotated examples can improve the trained model significantly. The semi-supervised mode was not part of the original Morfessor release, and was introduced by [Kohonen et al., 2010]. The main idea of this mode is to define two additional weight parameters, one for the likelihood of the unannotated data and one for the likelihood of the annotated data.

Morfessor FlatCat is a later addition of the Morfessor method family, that uses an HMM based architecture, supporting both unsupervised and semi-supervised learning, but cannot reach the score of Morfessor Categories-MAP [Grönroos et al., 2014].

One can find several publications that tried to improve Morfessor over the years. [Poon et al., 2009] for example uses a log-linear model and reduces F1 error by 11% as opposed to Morfessor Categories-MAP for the Arabic language. The base concept of the model is also MDL, but it differs from other methods in that it uses two priors, both favoring fewer morpheme types and fewer morpheme tokens during segmentation. The log-linear model can incorporate gold segmentation as well, and thus can be used in fully supervised and semi-supervised modes.

Lately, artificial intelligence is also frequently used in combination with Morfessor based models, such as in the proposed model of [Luong et al., 2013], where a context-sensitive morphological recurrent neural network was used alongside Morfessor.

MORSEL

MORSEL[3] [Lignos et al., 2009] is based on the base and transforms model originally published by Chan and Yang [Chan and Yang, 2008]. According to that, a word consists of a base and a transform. The base is not the stem, but the most frequent form of the morphologically related words. The transform defines the rule that needs to be applied to produce a derived form from the base. A transform is given as an affix pair (s_1, s_2) where s_1 needs to be removed from the base and s_2 needs to be appended.

During the training phase, the words of the training set are initially added to the set of unmodeled words, then they get processed one by one. A processed word can be moved to the set of bases or the set of derived words, while also storing the appropriate transform. This way, at the end of the training phase, the transforms will form word chains, starting from a base and then connecting one or multiple derived forms.

[2] `https://github.com/aalto-speech/morfessor`
[3] `https://github.com/ConstantineLignos/MORSEL`

DOI: 10.14750/ME.2022.020

The MORSEL method improves the original base and transforms model in several points, including:

- It can handle multiple decompositions of a word.
- It supports compounding. (Later improved in [Lignos, 2010].)
- It can determine how many learning iterations are needed.

Evaluation was executed using English and German data of MorphoChallenge 2009. In case of English, the results were better (higher F-measure of 58% and faster learning of 92 minutes), while in case of the German data set that contained more than 1.2 million records, learning took 375 minutes and the F-measure was only 33%.

In [Lignos, 2010], the MORSEL model was improved using better compounding strategies and introducing base inference, that means that even if a base is not included in the training set, the model infers it if it should exist based on the rest of the training data. This way, the results for German, Finnish and Turkish improved compared to the previous version.

MorphoChain

The MorphoChain method[4] [Narasimhan et al., 2015] incorporates semantic information as well into the segmentation learning process besides orthographical patterns.

From the training corpus, the MorphoChain model builds morphological chains that start from a base form and continue in inflected forms containing either prefixes or suffixes. Unlike Morfessor, MorphoChain also supports more complex transformations besides simple concatenative morphological rules. The different forms in a chain are stored in parent-child relationships, where the child is derived from the parent. Base forms do not have parents.

For the learning process, a log-linear model is used to predict the appropriate chains. The advantage of this log-linear model is that multiple features can be incorporated into the prediction. These features are represented by a feature vector $\phi : W \times Z \to \mathbb{R}^d$ and a corresponding weight vector $\theta \in \mathbb{R}^d$, where W is the set of words and Z is the set of candidates for words.

The MorphoChain model was evaluated using the Arabic, Turkish and Finnish languages, comparing it with other similar models including some Morfessor variants. Turkish and Finnish are agglutinative languages, and for them MorphoChain outperformed the other models by a small margin.

In [Bergmanis and Goldwater, 2017], an improved model is proposed that adapts the unsupervised MorphoChain system to provide morphological analysis that can abstract over spelling differences in functionally similar morphemes. Based on the presented test results, the proposed model outperforms both MorphoChain and MORSEL, and performs similarly to Morfessor.

2.2.5 SIGMORPHON

Lately, the state of the art morphology models are gathered by SIGMORPHON, the Special Interest Group on Computational Morphology and Phonology.[5] The training and test data is provided by them, and anyone can solve the announced tasks using any technique. The best achievements are summarized by their annual publication.

In this section I introduce those models that have been published as part of SIGMORPHON proceedings and also have their source code available online.

[4] `https://github.com/karthikncode/MorphoChain`
[5] `https://sigmorphon.github.io`

DOI: 10.14750/ME.2022.020

SIGMORPHON 2016

SIGMORPHON 2016 [Cotterell et al., 2016] was the last shared task publication that still included some models not applying artificial intelligence.

The Helsinki 2016 model[6] [Östling, 2016] used a one dimensional residual network architecture with constant size across layers, followed by either one or zero gated recurrent unit layer. The output vector of each residual layer is combined with the vector of the previous layer by addition, which means that the output is the sum of the input and the output of each layer.

Dropout was also used for regularization, with a dropout factor of 50%. The morphological features of the target form are concatenated to the 128-dimensional character embeddings at the top convolutional layer. Decoding is done by choosing the single most probable symbol at each letter position, according to the final softmax layer.

SIGMORPHON 2017

Since SIGMORPHON 2017 [Cotterell et al., 2017], the main emphasis of the shared tasks moved towards low-resource evaluation, and the resulting models all included some kind of AI based algorithms.

The UF 2017 method[7] [Zhu et al., 2017] models the morphological reinflection problem using an encoder-decoder architecture. For an input word, every character is encoded through a bidirectional gated recurrent unit network. Another GRU network is deployed as a decoder to generate the inflected word forms.

The UTNII 2017 model[8] [Senuma and Aizawa, 2017] is also based on the sequence to sequence model. In its basic form, the sequence to sequence model consists of two recurrent neural networks, the encoder and the decoder. After the encoder is fed with a sequence of input symbols, the hidden layer of the encoder is used as an input to the decoder, and finally the decoder emits a sequence of output symbols.

With the submitted configuration, UTNII 2017 was second in the high-resource scenarios, but unfortunately, in the medium-resource scenarios it only performed similarly to the baseline method, and in low-resource scenarios, it was the worst submitted model.

SIGMORPHON 2018

SIGMORPHON 2018 [Cotterell et al., 2018] increased the training and evaluation data size, and also brought in sentential context, which is out of the scope of this disseration.

The Hamburg 2018 model[9] [Schröder et al., 2018] introduces the concept of string transducer actions called patches. The resulting model is a language-agnostic network model that aims to reduce the number of learnt edit operations by introducing equivalence classes over graphical features of individual characters.

The IITBHU 2018 model[10] [Sharma et al., 2018] uses a pointer-generator network to mitigate the problem of copying many characters between word forms. This network architecture also helps in dealing with unknown characters. The lemma and the morphosyntactic tags are encoded by two separate encoders. While decoding,

---

[6] `https://github.com/robertostling/sigmorphon2016-system`
[7] `https://github.com/valdersoul/conll2017/tree/master/dl`
[8] `https://github.com/hajimes/conll2017-system`
[9] `https://gitlab.com/nats/sigmorphon18`
[10] `https://github.com/abhishek0318/conll-sigmorphon-2018`

DOI: 10.14750/ME.2022.020

the decoder reads relevant parts of the lemma and the tags using attention mechanism. Compared to other similarly performing systems, this model is trained end-to-end, does not require data augmentation techniques, and uses soft attention over hard monotonic attention, making the resulting system more flexible.

The MSU 2018 model[11] [Sorokin, 2018] aimed to improve the accuracy in medium and low resource scenarios by explicitly equipping the decoder with the information from the character-based language model, however the advantage was not clear.

Since 2019 [McCarthy et al., 2019], the focus of SIGMORPHON shifted to crosslingual tasks and contextual analysis using sentences, which are not the scope of this dissertation.

2.2.6 Morphological Analyzers for the Hungarian Language

Morphological analyzers exist for most languages, including Hungarian. In the following subsections I introduce four of the most popular tools.

Hunmorph-Ocamorph

Hunmorph-Ocamorph [Trón et al., 2005] is a pioneer morphological analyzer for the Hungarian language, that is part of the Szószablya project [Halácsy et al., 2003], developed by the Budapest University of Technology and Economics. The tool itself is called Hunmorph, but since there is an Ocamorph and a Foma based analyzer with the same name, I will call them Hunmorph-Ocamorph and Hunmorph-Foma, respectively.

The analyzer engine was designed in a way that is totally language independent. The dictionary for the Hungarian language is called Morphdb.hu [Trón et al., 2006], and stores lexical records with different affix flags. Both Hunmorph-Ocamorph and Morphdb.hu are open-source and can be downloaded from the project page.[12]

The output of Hunmorph-Ocamorph contains the lemma, its part of speech and the found affix type tokens in the word. For the word *tollakat*, which is the plural form and accusative case of the Hungarian word *toll* (*pen*), the output is:

$$\textit{toll/NOUN}\langle \textit{PLUR}\rangle\langle \textit{CAS}\langle \textit{ACC}\rangle\rangle$$

To preserve disk and memory space, the dictionary records contain general rules that cover the regular inflected forms and a list of irregular forms.

The transformation rules in the database have two main categories:

- Morphosyntactically active rules add something (a morpheme) to the root.
- Filter rules modify the previously existing form, changing for instance the vowel length.

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 in both directions, however, the Hunmorph-Ocamorph tool only supports morphological analysis.

Hunmorph-Foma

The foundation of the Hunmorph-Foma morphological analyzer is the Foma open-source finite-state toolkit [Hulden, 2009], which was originally the open-source implementation of the *lexc/xfst* grammatical analyzer of the Xerox laboratory.

[11] https://github.com/AlexeySorokin/Sigmorphon2018SharedTask
[12] http://mokk.bme.hu/en/resources/hunmorph

DOI: 10.14750/ME.2022.020

The main difference between Hunmorph-Foma and Hunmorph-Ocamorph other than their framework is the different affix type token systems they use. Therefore its output is slightly different, but contains the same pieces of information:

toll+Noun+Plur+Acc

Hunmorph-Foma is also open-source, and states in an introductory readme file on its Github page[13] that it is better than Hunmorph-Ocamorph, as it is based on a broader corpus, has lower memory consumption and is overall faster.

Humor

Humor [Prószéky and Tihanyi, 1993, Prószéky and Kis, 1999] (High-speed Unification MORphology) is a closed-source morphological analyzer implementation developed by MorphoLogic.[14] Unlike the other examined tools, only a DLL is present, the internal implementation details are not visible to us.

Humor is a rule based system, similarly to the previous two tools, but it was originally developed for Hungarian only, its engine is not language independent.

During the processing of an input word, the analyzer engine tries to break up the word to morphs, validates the connections of neighboring morphs and checks the validity of the full morphological 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.

Besides the morphological structure of the word, Humor also provides the affix boundaries:

toll[FN]+ak[PL]+at[ACC]

If the lexical and surface forms are different, the tool also returns both of them.

[Prószéky and Novák, 2005] used the formalism of Humor and an improved, non-redundant database format to create a lemmatizer and a morphological generator for Uralic languages.

Hunspell

Hunspell[15] is an open-source spell checker of popular applications like LibreOffice, OpenOffice.org, Mozilla Firefox, Mozilla Thunderbird and Google Chrome. Its base framework was published in [Pirinen et al., 2010]. Although it is mainly used for spell checking, it has other use cases as well, including morphological analysis.

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

The output of Hunspell is much different than the previous three tools:

tollakat st:toll po:noun ts:PLUR ts:NOM is:PLUR is:ACC

[13] `https://github.com/r0ller/hunmorph-foma`
[14] `https://www.morphologic.hu`
[15] `https://hunspell.github.io`

DOI: 10.14750/ME.2022.020

2.3 Conclusion

In this chapter I summarized the main morphology models that can be found in literature, and categorized them according to multiple dimensions, including:

- Knowledge representation: dictionary based, rule based, statistical and AI based models
- Scope: single-affix and multi-affix models
- Symmetry: asymmetric and symmetric models
- Granularity of analysis: morphological analysis, segmentation, lemmatization and stemming models
- Machine learning capabilities: non-automated, supervised, unsupervised and semi-supervised training

Some of the models were explored in details as part of separate sections:

- Two-level morphology in Subsection 2.2.1
- FSTs in Subsection 2.2.2
- TASR in Subsection 2.2.3
- Morfessor, MORSEL and MorphoChain in Subsection 2.2.4
- SIGMORPHON models (Helsinki 2016, UF 2017, UTNII 2017, Hamburg 2018, IITBHU 2018, MSU 2018) in Subsection 2.2.5
- Morphological analyzers for the Hungarian language including Hunmorph-Ocamorph, Hunmorph-Foma, Humor and Hunspell in Subsection 2.2.6

Those models that were examined in their own sections will be used as baseline models in the upcoming chapters (except for two-level morphology). Table 2.1 contains these models according to their main characteristics. An asterisk (*) marks the unsupervised and semi-supervised models.

TABLE 2.1: Categorization of the baseline morphology models

		Dictionary		Rule		Statistical & AI	
		Single	Multi	Single	Multi	Single	Multi
Infl.	Asymm.	FST		TASR	Two-Level		SIGMORPHON*
	Symm.				*Target*		
Analysis	Asymm.				Hunmorph Humor Hunspell		
	Symm.				*Target*		
Segm.	Asymm.				Porter		Morfessor* MORSEL* MorphoChain*
	Symm.						

As we can see, FSTs and TASR are single-affix asymmetric inflection models, while the analyzers are multi-affix asymmetric analysis tools. FSTs store the content of dictionaries in a compact way, while the others are rule based systems. On the other hand, Morfessor, MORSEL and MorphoChain are unsupervised multi-affix asymmetric statistical segmentation models, and the SIGMORPHON models are used for inflection generation.

The goal of my research is to propose a novel morphology model that can solve both the inflection generation and morphological analysis problems in a multi-affix,

DOI: 10.14750/ME.2022.020

symmetric manner. My goal is also to create a rule model that can be efficiently applied for agglutinative languages like Hungarian.

In Chapter 4 I will propose efficient single-affix transformation engine models. Then in Chapter 5 I will propose a higher-level multi-affix model that will be able to manage affix type chains as well in a multi-affix environment, handling all the affix types of the target language.

DOI: 10.14750/ME.2022.020

Chapter 3

The Analysis of Existing Hungarian Morphological Analyzers

Since the target language of my research is Hungarian, in this chapter I analyze four popular morphological analyzers for the Hungarian language:

- Hunmorph-Ocamorph [Trón et al., 2005, Trón et al., 2006],[1]
- Hunmorph-Foma,[2]
- Humor [Prószéky and Tihanyi, 1993, Prószéky and Kis, 1999] and
- Hunspell.[3]

The main question that needs to be answered is which one is worth using as a baseline method, which one is the *best* among them.

It is important to note that only a few similar publications can be found in literature that compare Hungarian analyzers with each other, but in most cases, these publications only examine the stemming accuracy of the models.

For example, [Endrédy, 2015] calculates an error value for every word in the test corpus, using overstemming and understemming indices. Another paper compares the above four analyzers [Endrédy and Novák, 2015], as well as the Porter stemmer [Porter, 1980], its Hungarian adaptation [Porter, 2001, Tordai and de Rijke, 2006] and some Apache Lucene [McCandless et al., 2010] modules like KStem, Porter, English-Minimal, Stempel and Morfologik. The used metrics include 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.

In contrast, my goal is to also examine the token systems of the morphological analyzers, and how well they can analyze the input words. To remain objective, in Section 3.1 I introduce some formulae that I will use during the comparison of the four analyzers and that are based on measurable values regarding the analyzers and their annotation token systems, including:

1. What are the main differences of the analyzer outputs and token systems? Are there morphosyntactic tags that are only recognized by one analyzer and not others, or most of these tags are present in the token system of all the analyzers?
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?

[1] http://mokk.bme.hu/en/resources/hunmorph

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

[3] https://hunspell.github.io

DOI: 10.14750/ME.2022.020

I introduce similarity and distance formulae for the examined morphological analyzers. The distances are also visualized in 2D Euclidean space. I use a Hungarian corpus collected from the Internet for the analysis. The automated generation process of this corpus is described later in Section 7.1.

The analysis of the measured values can be found in Section 3.2. Section 3.3 summarizes the results of this chapter.

3.1 Similarity and Distance of Morphological Analyzers

Formally, let W denote the set of words in the test corpus, $w \in W$. The set of lemmas is a subset of the set of words: $\bar{w} \in \bar{W} \subset W$.

A morphological analyzer is formally a mapping denoted by A, that maps words to pairs containing a lemma and a list of morphosyntactic tags:

$$A : W \to \left\{ \left(\bar{w}, \left\langle T^A \right\rangle \right) \right\} \tag{3.1}$$

The set of all the morphosyntactic tags related to the analyzer A will be denoted by $\mathbb{T}^A$. For convenience, $l\left(A\left(w\right)\right)$ will denote the set of all the possible lemmas of the input word, while $t\left(A\left(w\right)\right)$ will denote the set of all the possible token lists provided by the morphological analyzer A for the input word w.

The set of recognized words by the morphological analyzer A is denoted by

$$W^A = \{ w \in W \mid |A\left(w\right)| > 0 \} \tag{3.2}$$

We can also calculate the recognition ratio of W^A in the whole corpus:

$$\nu^A = \frac{\left|W^A\right|}{|W|} \tag{3.3}$$

The recognition similarity is based on the number of recognized words of a morphological analyzer pair:

$$S^R_{A_i,A_j} = \frac{\left|W^{A_i} \cap W^{A_j}\right|}{\left|W^{A_i} \cup W^{A_j}\right|} \tag{3.4}$$

This can easily be converted to a distance value between the two morphological analyzers:

$$D^R_{A_i,A_j} = \frac{1}{S^R_{A_i,A_j}} \tag{3.5}$$

Remark 3.1 (The limits of the recognition similarity and distance)**.** *From Equation 3.4 we can see that* $0 \leq S^R_{A_i,A_j} \leq 1$. *If* $W^{A_i} \cap W^{A_j} = \emptyset$, *then the similarity will be* 0, *while if* $W^{A_i} = W^{A_j}$, *then the similarity will be* 1.

Since the recognition distance is the multiplicative inverse of the recognition similarity, its value is between 1 *and infinity. The minimum value occurs if the similarity is* 1.

From this remark, we can see that the recognition distance (and the forthcoming distance measurements) are not real distances, they only point out the similarities and differences of morphological analyzers. However, since in practice we do not tend to compare identical analyzers (and no two morphological analyzers can be treated equal), this does not mean a real problem. If we still wanted to mitigate this issue, we could simply subtract 1 from the distance value.

To measure the similarity of the annotation token systems and equivalence of the examined morphological analyzers, we first need to define a mapping among

DOI: 10.14750/ME.2022.020

the annotation token systems. The mapping function that converts the annotation tokens of A_i to those of A_j is denoted by m_{A_i,A_j}. Using such a mapping, it is easy to convert morphosyntactic tags in a list, one by one:

$$m_{A_i,A_j}\left(\left\langle T_1^{A_i},\ldots,T_n^{A_i}\right\rangle\right)=\left\langle m_{A_i,A_j}\left(T_1^{A_i}\right),\ldots,m_{A_i,A_j}\left(T_n^{A_i}\right)\right\rangle \tag{3.6}$$

Note that $\langle\rangle$ denotes an ordered list of elements here.

The mapping m_{A_i,A_j} operator is perfect if for every word that is recognized by both analyzers, there is at least one output pair provided by A_i and one output pair provided by A_j for which the given lemmas are the same and the mapping maps the token list provided by A_i to the token list provided by A_j. Usually it is impossible to define a perfect mapping between the annotation tokens of two morphological analyzers, since there might be tokens in one system that does not exist in the other one. For the evaluation of the four analyzers, I created a mapping that can be found in Appendix A.

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

$$S^T_{A_i,A_j,m_{A_i,A_j}}=\frac{\left|\left\{T^{A_i}\mid\exists T^{A_j}:\ m_{A_i,A_j}\left(T^{A_i}\right)=T^{A_j}\right\}\right|}{\left|\mathbb{T}^{A_i}\right|} \tag{3.7}$$

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

$$D^T_{A_i,A_j,m_{A_i,A_j}}=\frac{1}{S^T_{A_i,A_j,m_{A_i,A_j}}} \tag{3.8}$$

The mapping similarity denotes how many words result in equivalent annotation tokens between two morphological analyzers based on a mapping:

$$S^M_{A_i,A_j,m_{A_i,A_j}}=\frac{\left|\left\{w\in W^{A_i}\cup W^{A_j}\mid\left|m_{A_i,A_j}\left(t\left(A_i\left(w\right)\right)\right)\cap t\left(A_j\left(w\right)\right)\right|>0\right\}\right|}{\left|W^{A_i}\cup W^{A_j}\right|} \tag{3.9}$$

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

$$D^M_{A_i,A_j,m_{A_i,A_j}}=\frac{1}{S^M_{A_i,A_j,m_{A_i,A_j}}} \tag{3.10}$$

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

$$D^C_{A_i,A_j,m_{A_i,A_j}}=D^R_{A_i,A_j}+D^T_{A_i,A_j,m_{A_i,A_j}}+D^M_{A_i,A_j,m_{A_i,A_j}} \tag{3.11}$$

3.2 Analyzing the Similarities and Differences of the Morphological Analyzers

In this section, I present the calculated values of the recognition, token and mapping similarity, as well as the recognition, token, mapping and cumulative distance. The distances will also be visualized using 2D Cartesian coordinate systems. These coordinate systems were generated using a dimension reduction based projection technique. Therefore the coordinates of the points in the diagrams are not exact, they are only approximations that reflect the calculated distance values.

DOI: 10.14750/ME.2022.020

The data source of the calculation is a Hungarian corpus that was generated from Hungarian free texts collected from the Internet, as described later in Section 7.1. The mapping used during the measurements is the one provided in Appendix A.

3.2.1 Comparison of the Annotation Token Systems

Table 3.1 contains the token similarity and distance values among the examined morphological analyzers, using the notations of the previous section.

TABLE 3.1: Token similarity $\left(S^T_{A_i,A_j,m_{A_i,A_j}}\right)$ and distance $\left(D^T_{A_i,A_j,m_{A_i,A_j}}\right)$ values among the morphological analyzers

A_i	A_j	$S^T_{A_i,A_j,m_{A_i,A_j}}$	$D^T_{A_i,A_j,m_{A_i,A_j}}$
Hunmorph-Ocamorph	Hunmorph-Foma	0.4855	2.0598
Hunmorph-Ocamorph	Humor	0.4309	2.3208
Hunmorph-Ocamorph	Hunspell	0.5749	1.7394
Hunmorph-Foma	Humor	0.5253	1.9036
Hunmorph-Foma	Hunspell	0.5856	1.7075
Humor	Hunspell	0.4623	2.1630

Figure 3.1 visualizes the approximation of these distance values in 2D space. From this image, we can see that the token system of the four analyzers are different from each other, since they all have approximately the same distance from each other.

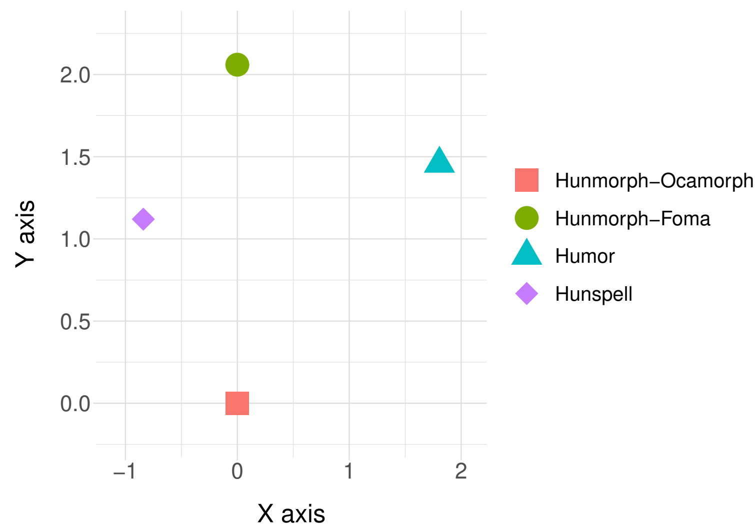


FIGURE 3.1: Visualizing the annotation token system distances

3.2.2 Recognition Statistics

Table 3.2 contains the number of recognized words by each examined morphological analyzer, and their recognition ratios. The total number of unique words was 3,852,592. From the table, it can be seen that Hunmorph-Foma recognized the most

DOI: 10.14750/ME.2022.020

TABLE 3.2: The number of recognized words $\left(\left|W^{A_i}\right|\right)$ and the ratio of recognized words $\left(\nu^{A_i}\right)$ for each morphological analyzer

A_i	$\left\|W^{A_i}\right\|$	ν^{A_i}
Hunmorph-Ocamorph	2,515,570	65.30%
Hunmorph-Foma	3,154,529	81.88%
Humor	823,531	21.38%
Hunspell	99,441	2.58%

words from the test corpus, while Humor and Hunspell performed significantly worse.

We can also examine how many words are recognized by only one morphological analyzer, by morphological analyzer pairs, three morphological analyzers and all of the examined morphological analyzers. First, Table 3.3 contains the number of words recognized by only one morphological analyzer.

Hunspell has the lowest number here as well, meaning that the number of words that are only recognized by Hunspell and not the other analyzers is the lowest.

TABLE 3.3: The number of words recognized by only one morphological analyzer

A	**Words recognized by only A**
Hunmorph-Ocamorph	159,651
Hunmorph-Foma	723,968
Humor	387,380
Hunspell	16,256

In Table 3.4, we can see how many commonly recognized words there are between the morphological analyzer pairs. As we can see, Hunmorph-Ocamorph and Hunmorph-Foma are the most similar to each other, while Hunspell is the most different from the other three analyzers.

TABLE 3.4: The number of words recognized by exactly two morphological analyzers

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

Table 3.5 contains the commonly recognized words among exactly three morphological analyzers.

It is not surprising that the first row has the highest value, since Hunmorph-Ocamorph and Hunmorph-Foma are the most similar to each other, while Hunspell has the least common elements with the other models.

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

Figure 3.2 display the Venn diagram of the three strongest morphological analyzers, namely Hunmorph-Ocamorph, Hunmorph-Foma and Humor.

DOI: 10.14750/ME.2022.020

TABLE 3.5: The number of words recognized by exactly three morphological analyzers

Morphological analyzers	Recognized words
Hunmorph-Ocamorph, Hunmorph-Foma, Humor	336,339
Hunmorph-Ocamorph, Hunmorph-Foma, Hunspell	48,200
Hunmorph-Ocamorph, Humor, Hunspell	99
Hunmorph-Foma, Humor, Hunspell	1,654

Based on the above statistics, Hunmorph-Foma seemed to be the best alternative, but after some detailed examination, I found that in some cases it could not return the valid lemma. A simple example is the word *(merengők, mereng+Adj+Plur)* which is the plural of *meditative* in Hungarian. According to the morphological structure provided by Hunmorph-Foma, the lemma is *mereng* (*meditate*) which is in reality a verb, but according to the provided annotation tokens, it is an adjective, which is incorrect.

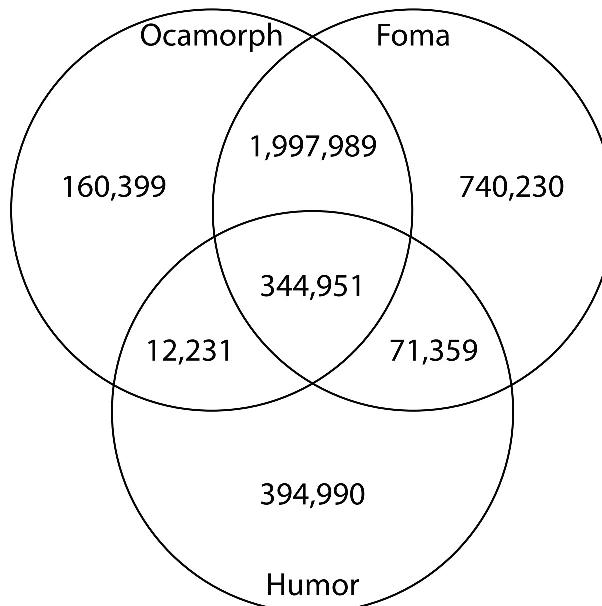


FIGURE 3.2: Venn diagram of the number of recognized words for the three strongest morphological analyzers

Another problem with Hunmorph-Foma was the handling of adjective derivation. A sample response 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*). Hunmorph-Ocamorph can recognize this fact and produce the correct lemma.

Table 3.6 shows the recognition similarity and distance values for all the examined morphological analyzer pairs.

Figure 3.3 visualizes the approximation of these distance values. Hunmorph-Ocamorph and Hunmorph-Foma are very close to each other, while Hunspell is very far away from the other three morphological analyzers.

DOI: 10.14750/ME.2022.020

TABLE 3.6: Recognition similarity $\left(S^{R}_{A_i,A_j}\right)$ and distance $\left(D^{R}_{A_i,A_j}\right)$ values among the morphological analyzers

A_i	A_j	$S^{R}_{A_i,A_j}$	$D^{R}_{A_i,A_j}$
Hunmorph-Ocamorph	Hunmorph-Foma	0.4132	2.4201
Hunmorph-Ocamorph	Humor	0.1070	9.3485
Hunmorph-Ocamorph	Hunspell	0.0220	45.3530
Hunmorph-Foma	Humor	0.0898	11.1373
Hunmorph-Foma	Hunspell	0.0177	56.4347
Humor	Hunspell	0.0195	51.3475

FIGURE 3.3: Visualizing the recognition distances

3.2.3 Mapping Among the Examined Morphological Analyzers

The mapping similarity and distance values among the morphological analyzer pairs can be seen in Table 3.7.

TABLE 3.7: Mapping similarity $\left(S^{M}_{A_i,A_j,m_{A_i,A_j}}\right)$ and distance $\left(D^{M}_{A_i,A_j,m_{A_i,A_j}}\right)$ values among the morphological analyzers

A_i	A_j	$S^{M}_{A_i,A_j,m_{A_i,A_j}}$	$D^{M}_{A_i,A_j,m_{A_i,A_j}}$
Hunmorph-Ocamorph	Hunmorph-Foma	0.7759	1.2887
Hunmorph-Ocamorph	Humor	0.9188	1.0883
Hunmorph-Ocamorph	Hunspell	0.7969	1.2549
Hunmorph-Foma	Humor	0.8727	1.1458
Hunmorph-Foma	Hunspell	0.4497	2.2237
Humor	Hunspell	0.7372	1.3564

Figure 3.4 displays these distance values in 2D space. The reason why the analyzers are far from each other is that Humor and Hunspell recognize far less words

DOI: 10.14750/ME.2022.020

than Hunmorph-Ocamorph and Hunmorph-Foma, while Hunmorph-Foma has a couple of problems as described in the previous subsection.

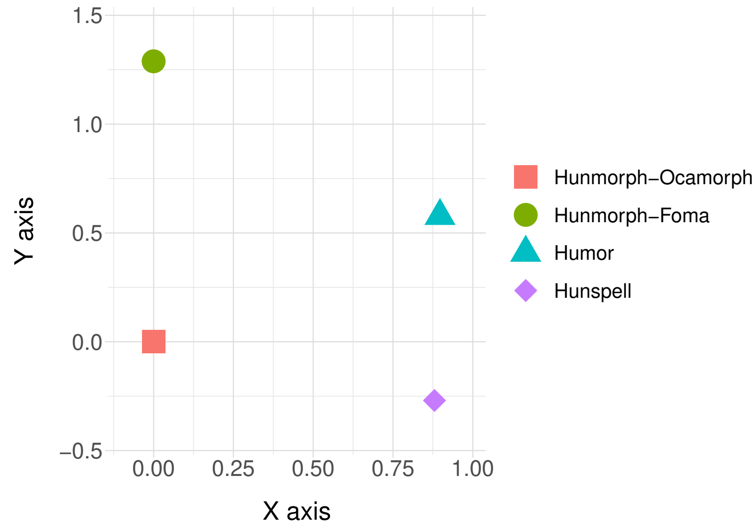


FIGURE 3.4: Visualizing the mapping distances

3.2.4 Cumulative Distance

The cumulative distance is the sum of the recognition, token and mapping distances. Figure 3.5 displays the cumulative distances in 2D space, while the exact distance values can be seen in Table 3.8.

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.

Since Hunmorph-Ocamorph and Hunmorph-Foma recognized the most words from the data set and they are closest to each other regarding the cumulative distance, they seem to be the best morphological analyzers among the four tools. Considering the problems of Hunmorph-Foma described earlier, I chose Hunmorph-Ocamorph as the baseline model for the proposed models described in later chapters.

TABLE 3.8: Cumulative distance $\left(D^{C}_{A_i,A_j,m_{A_i,A_j}}\right)$ values among the morphological analyzers

A_i	A_j	$D^{C}_{A_i,A_j,m_{A_i,A_j}}$
Hunmorph-Ocamorph	Hunmorph-Foma	5.7686
Hunmorph-Ocamorph	Humor	12.7576
Hunmorph-Ocamorph	Hunspell	48.3473
Hunmorph-Foma	Humor	14.1867
Hunmorph-Foma	Hunspell	60.3659
Humor	Hunspell	54.8669

DOI: 10.14750/ME.2022.020

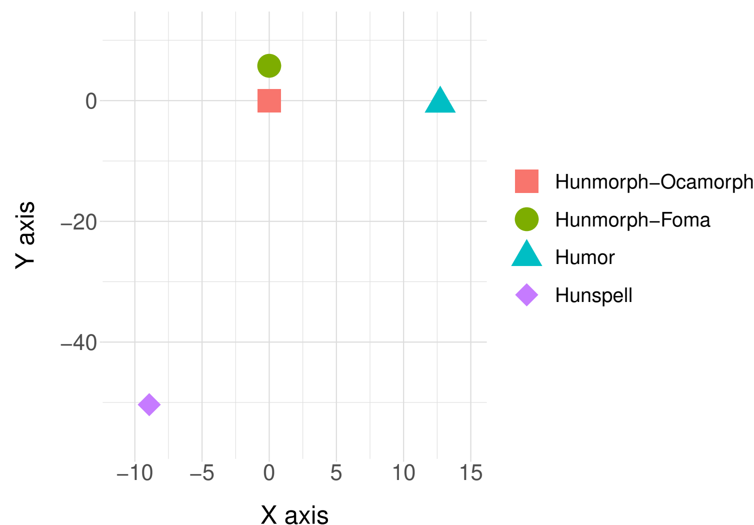


FIGURE 3.5: Visualizing the cumulative distances

3.3 Conclusion

In this chapter I analyzed four popular morphological analyzers for the Hungarian language, namely Hunmorph-Ocamorph, Hunmorph-Foma, Humor and Hunspell. The main question that I wanted to answer is which one of them is the *best*, which one is worth using in later chapters as a baseline model.

To remain objective, I introduced several formulae: the recognition similarity that measures the number of recognized words, the token similarity that measures the similarity of the annotation token systems of these analyzers, and finally the mapping similarity that measures how many words are analyzed equivalently by the analyzers. Each similarity value can be converted to a distance value: the recognition, token and mapping distance. Finally, I also introduced a cumulative distance that incorporates all the above distance values into a single measure.

From the tables and figures of the previous section we can see that Hunmorph-Ocamorph and Hunmorph-Foma are the two best morphological analyzers, as both Humor and Hunspell are far from them regarding all of the distance values. Although Hunmorph-Foma seems better in some respects, there are several words in the test corpus that it cannot handle correctly. Therefore I chose Hunmorph-Ocamorph to use in later chapters.

Thesis 1 **[1]**

I have designed and implemented a new method to compare, analyze, evaluate and rank morphological analyzers. This method is based on novel formulae to calculate similarity and distance values among the different analyzers, including the recognition similarity, token similarity, mapping similarity; as well as the recognition distance, token distance, mapping distance and cumulative distance. I applied this analysis method on four popular morphological analyzers of the Hungarian language, namely Hunmorph-Ocamorph, Hunmorph-Foma, Humor and Hunspell. For the evaluation, I created a token mapping among these analyzers as well. Based on the performed evaluation, Hunmorph-Ocamorph proved to be the most usable model among the four analyzers.

DOI: 10.14750/ME.2022.020

Chapter 4

Single-Affix Transformation Engine Model

In this chapter I propose two novel single-affix transformation engine models that can learn inflection rules from a provided training word pair set to solve the single-affix inflection generation problem.

The first model, presented in Section 4.1 stores its rules in a lattice structure. Its rule model is more complex, containing abstract transformation steps and indices that identify the changing position in the words. During the training phase, an improved Levenshtein distance based algorithm is used to identify the changing substrings. To build this lattice, I propose 3 different builder algorithms.

The second model called ASTRA (Atomic String Transformation Rule Assembler), presented in Section 4.2 has a simpler rule model that focuses on simple string transformations. Besides the transformation context, the rules contain only a changing substring and a replacement string. These rules are stored in a set or a prefix tree. The advantage of ASTRA over the lattice based model is that it is truly symmetric, thus its rules can be used during both inflection generation and morphological analysis. Moreover, its rule generation process is faster due to the simpler storage structure.

Section 4.3 contains all the performed experiments. In this section I evaluate the average training and search time, the average size and the average accuracy of all the presented model variants, comparing these metrics with those of a simple dictionary, an FST and the TASR model. As we will see, the generalization capability of ASTRA is outstanding among these models.

4.1 Lattice Based Model

The main concept behind the proposed lattice based transformation engine model is to treat inflection generation as a classification problem. The transformation rules extracted from the training data set contain both the context of the grammatical transformations and their elementary transformation steps. This way, if we find a transformation rule in the knowledge base whose context is close to the input word, we can apply its transformation steps on the input to generate the inflected form.

These transformation rules are stored in a lattice structure based on parent-child relationships. Storing the rules in a lattice has several benefits: it speeds up the search time later and makes it easier to gather the most general transformation rules of the target affix type. For example we can easily see from a generated lattice that the most general rule of the Hungarian accusative case is to insert a *'t'* character at the end of the word.

DOI: 10.14750/ME.2022.020

4.1.1 The Theory of Formal Concept Analysis

The roots of formal concept analysis (FCA) [Ganter and Wille, 2012] originate from the applied lattice and order theory [Birkhoff, 1940], as well as the theory of Galois connections [Ore, 1944].

Binary relations and set theory form the basis of FCA [Grätzer, 2003]. A binary relation $\leq$ on the set M is a set of pairs (m, n) where $m, n \in M$. The $\leq$ relation is called an order relation if it is

- reflexive: $m \leq m$,
- antisymmetric: if $m \leq n$ and $n \leq m$ then $m = n$, and
- transitive: if $m \leq n$ and $n \leq o$ then $m \leq o$.

An ordered set is a pair $(M, \leq)$ where M is a set and $\leq$ is an order relation.

An important concept in lattice theory is the neighborhood of nodes. If $m, n \in M$ and $m < n$ while $\nexists o \in M$ such that $m < o < n$, then m is called the lower neighbor of n, and n is called the upper neighbor of m.

Let $(M, \leq)$ be an ordered set and $A \subseteq M$. In this case, $m \in M$ is a lower (or upper) bound of A if for every $a \in A$, $m \leq a$ (or $m \geq a$). Among all the lower bounds, the largest is called the infimum (denoted by $\bigwedge A$), while among all the upper bounds, the smallest one is called the supremum (denoted by $\bigvee A$).

Example 4.1 (Infimum and supremum)**.** *If M is the lattice in Figure 4.1 and $A \subseteq M$ contains the nodes a, b, c, e, f and g, then a and 1 are upper bounds of A, a being its supremum. Likewise, 0 is a lower bound and the infimum of A.*

Definition 4.1 (Lattice)**.** *An ordered set $\mathbf{M} = (M, \leq)$ is a lattice if any two elements $m, n \in M$ has a supremum $m \vee n$ and an infimum $m \wedge n$.*

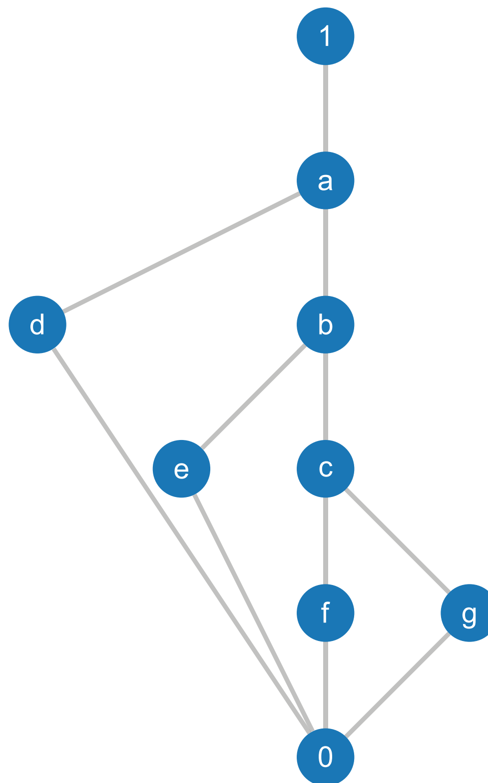


FIGURE 4.1: A sample lattice

DOI: 10.14750/ME.2022.020

Lattices can be visualized using Hasse diagrams, where each node is represented by a circle, and neighbors are connected with lines. Upper neighbors are displayed higher, lower neighbors are displayed lower. One such lattice can be seen in Figure 4.1.

Definition 4.2 (Complete lattice). ***M*** *is a complete lattice if for any* $A \subseteq M$ *subset, the supremum* $\bigvee A$ *and the infimum* $\bigwedge A$ *exist.*

In a complete lattice, $\bigvee M$ is called the unit element (denoted by 1), while $\bigwedge M$ is called the zero element (denoted by 0). The upper neighbors of the zero node are called atoms, while the lower neighbors of the unit node are called coatoms.

A chain in a lattice contains nodes that are all comparable with each other, i.e. it is a subset $A \subseteq M$ where $m \leq n$ or $m \geq n$, $\forall m, n \in A$. Those nodes that are incomparable with each other form an antichain. The size of a lattice can be defined using its width (the size of its maximal antichain) and height (the size of its longest chain minus one).

Example 4.2 (Lattice width and height). *The lattice in Figure 4.1 has a width of 4 and a height of 5. One of its longest chain contains the nodes* 1*,* a*,* b*,* c*,* f *and* 0*, while its longest antichain contains* d*,* e*,* f *and* g*.*

In FCA, the set of attributes common to the objects in $A \subseteq G$ is defined as $A' = \{m \in M \mid gIm, \forall g \in A\}$. Similarly, the set of objects which have all the attributes in $B \subseteq M$ is defined as $B' = \{g \in G \mid gIm, \forall m \in B\}$.

Definition 4.3 (Formal concept). (A, B) *is a formal concept of the context* (G, M, I) *if* $A \subseteq G$*,* $B \subseteq M$*,* $A' = B$ *and* $B' = A$ *where* G *is a set of objects,* M *is a set of attributes,* I *is the relation between objects and attributes,* A *is called the extent and* B *is called the intent of the concept* (A, B)*.*

Definition 4.4 (Concept lattice). *The concept lattice of a formal context* (G, M, I) *is the set of all formal concepts of* (G, M, I)*, together with the partial order* $(A_1, B_1) \leq (A_2, B_2) \Leftrightarrow A_1 \subseteq A_2 \Leftrightarrow B_1 \supseteq B_2$*.*

One of the first proposals that applied concept lattice theory to solve the classification problem is [Zhao and Yao, 2006]. In this model, one of the attributes is marked as the class label. A classification rule describes the dependency among the class label and the other attributes. Each classification rule has a confidence value between 0 and 1. The higher the confidence of a rule is, the more accurate the rule is. If the confidence value is 1, the rule is said to be consistent. It can be seen that the confidence value of an ancestor rule cannot be higher than any of its descendants. The goal of the model is to find the most general matching consistent concept.

4.1.2 Levenshtein Distance Based Transformation Rule Generation

To identify the changing substrings in the base words and the necessary steps to form the inflected word forms, I use the Levenshtein distance [Levenshtein, 1966] or edit distance that helps in quantifying the difference, and minimize the required edit steps between two strings.

Unit Cost Model for Levenshtein Distance Calculation

The base concept of using the Levenshtein distance to identify the changing substrings is that we can generate elementary transformation steps (character additions,

DOI: 10.14750/ME.2022.020

deletions and replacements) that yield the inflected word form from the base form. Each transformation step has a cost, and the sum of these costs needs to be minimized. The goal is to find the transformation rule having minimal cost.

Let Σ be the alphabet of non-empty characters, $c \in \Sigma$. The empty character will be denoted by $\emptyset$. A transformation step can be formalized as

$$\delta^L \in \Sigma \cup \{\emptyset\} \times \Sigma \cup \{\emptyset\} \tag{4.1}$$

Transformation steps have four different categories: character addition, character removal, invariant replacement (the character is left as is) and character replacement. These categories will be denoted as:

$$\begin{aligned} \delta^L_+ &= (\emptyset, c) \\ \delta^L_- &= (c, \emptyset) \\ \delta^L_= &= (c, c) \\ \delta^L_{\neq} &= \left(c, c'\right) \end{aligned} \tag{4.2}$$

where $c, c' \in \Sigma$ are arbitrary characters, $c \neq c'$.

These elementary transformation steps can be organized into a transformation path $\Delta^L = \left\langle \delta^L_i \right\rangle$ that shows how to transform the base form into the inflected form.[1] Each transformation step has an associated cost on this path ($cost\left(\delta^L\right)$). According to the original Levenshtein distance calculation formulae, only the invariant replacement has a 0 cost, all the others have a cost of 1:

$$\begin{aligned} cost\left(\delta^L_=\right) &= 0 \\ cost\left(\delta^L_+\right) = cost\left(\delta^L_-\right) = cost\left(\delta^L_{\neq}\right) &= 1 \end{aligned} \tag{4.3}$$

The total cost of the path is the sum of the costs of its steps, this is the metric that needs to be minimized:

$$\min_{\Delta^L} \sum_{\delta^L \in \Delta^L} cost\left(\delta^L\right) \tag{4.4}$$

Example 4.3 (Levenshtein matrix)**.** *In Figure 4.2 we can see two optimal Levenshtein matrices for the Hungarian word pair (alma, almát), the base form and accusative case of apple in Hungarian.*

On the left side, the first 3 characters are left untouched, then the 'a' is replaced with an 'á', finally an extra 't' character is appended to the word. On the right side, we can see that the same cost can be achieved by first inserting the 'á' character, then replacing the 'a' with a 't'.

As we can see in Figure 4.2, the original cost function may lead to several transformation paths for which the total cost is optimal (in this case, the optimal cost is 2). Although the algorithm cannot distinguish among these solutions, Hungarian speakers will likely to choose the left matrix as the correct one.

Improved Cost Function

To reduce the number of optimal transformation paths, a new cost function is introduced that results in a more fine-grained cost distribution. The intuition behind the improved cost function is that historically the grammatical rules of the Hungarian

[1] Throughout the dissertation, the $\langle x_i \rangle_{i=1}^n$ formalism means a list of elements where the order of elements is fixed. If the ordering is not defined, the usual $\{x_i\}_{i=1}^n$ set formalism is used.

DOI: 10.14750/ME.2022.020

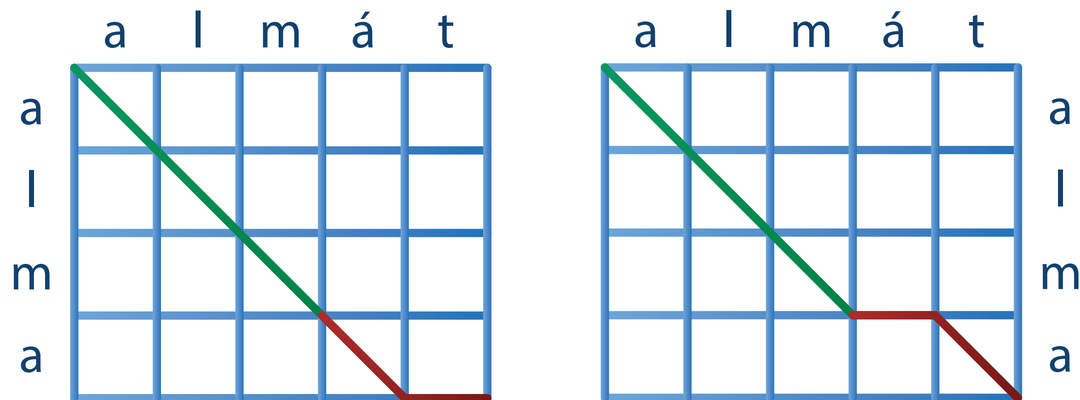


FIGURE 4.2: Two possible Levenshtein matrices with optimal cost for the Hungarian word pair *(alma, almát)*

language had been formed according to phonetic attributes, i.e. those rules are more usual that do not require us to pronounce sounds that are phonetically far from each other.

TABLE 4.1: Hungarian vowel attributes

		Horizontal Tongue Position							
		Front				Back			
Vertical Tongue Position	Close	ü	ű	i	í	u	ú		
	Middle	ö	ő		*(é)*	o	ó		
	Semi-Open			e	*é**	a	*á**		
	Open								*(á)*
Length		Short	Long	Short	Long	Short	Long	Short	Long
Lip Shape		Rounded		Unrounded		Rounded		Unrounded	

The Hungarian language has four main phonetic attributes for vowels (Table 4.1) and four for consonants (Table 4.2). In case of vowels, I made a minor change that I marked with asterisks (*), namely the characters *'á'* and *'é'* were moved next to their short counterparts. The abbreviations in case of consonant attributes in Table 4.2 are the following: Fricative (Fric), Laterial-Fricative (Lat-Fric), Lateral-Approximative (Lat-App), Voiced (V), Unvoiced (U), Labio-Dental (Labio-Dent) and Dental-Alveolar (Dent-Alv).

Based on these phonetic attributes, the improved cost function will return the number of changing attribute values. For example, the cost of a character insertion or deletion will be equal to the number of attributes in the inserted or deleted character, but in case of replacement, the cost will be often less than the total number of attributes.

Example 4.4 (Improved Levenshtein cost function)**.** *Returning to Figure 4.2, we can see that using the new, improved cost function, replacing an 'a' character with 'á' only changes the length of the vowel, so the cost of this step is* 1*. On the other hand, replacing 'a' with 't' will change every attribute, making the cost equal to* 8*. Inserting an 'á' or a 't' character*

DOI: 10.14750/ME.2022.020

TABLE 4.2: Hungarian consonant attributes

		Way of production								
		Plosive			Fric		Lat-Fric		Lat-App	Trill
	Voice	V		U	V	U	V	U	V	V
Place of production	Bilabial	m	b	p						
	Labio-Dent				v	f				
	Dent-Alv	n	d	t	z	sz	dz	c	l	r
	Dental-Postalveolar				zs	s	dzs	cs		
	Palatal	ny	gy	ty	j					
	Velar		g	k						
	Glottal					h				
Position of uvula		Nasal	Oral							

has a cost of 4*, so the total cost of the right matrix is* $4 + 8 = 12$*, while the total cost of the left matrix is* $1 + 4 = 5$*. This means that the new cost function will be able to distinguish between these two cases and will correctly choose the intuitive solution.*

4.1.3 The Lattice Rule Model

The transformation rules extracted from the training word pair set have the following structure:

$$R^L = \left(\alpha^L, \sigma^L, \omega^L, \eta_f^L, \eta_b^L, \Delta^L\right) \tag{4.5}$$

where

- $\alpha^L \in \Sigma^*$ is the prefix of the rule, containing the characters before the changing substring,
- $\sigma^L \in \Sigma^*$ is the core of the rule, i.e. the changing substring,
- $\omega^L \in \Sigma^*$ is the postfix of the rule, containing the characters after the changing substring,
- η_f^L is the front index of the rule context occurrence in the source word from its beginning (i.e. if $\eta_f^L = 1$ then the first occurence of the context must be transformed from the beginning of the source word),
- η_b^L is the back index of the rule context occurrence in the source word from its end (i.e. if $\eta_b^L = 1$ then the first occurence of the context must be transformed from the end of the source word), and
- $\Delta^L = \langle \delta_i^L \rangle$ is a list of elementary transformation steps on the core, $\delta_i^L \subseteq \Sigma \cup \{\emptyset\} \times \Sigma \cup \{\emptyset\}$.

The context of a rule is the concatenation of its prefix, core and postfix components: $\gamma\left(R^L\right) = \alpha^L + \sigma^L + \omega^L$.

Example 4.5 (Lattice rules)**.** *Let us have (xabyx**ab**yz, xabyx**cdw**yz) as an artificial word pair. We can create multiple rules that will cover this word pair, two of them can be seen in Table 4.3.*

To build a complete lattice from the generated rules that can also generalize, we need to also generate all the rule intersections. The parent-child relationship among the rules is defined by the `IsSubsetOf` operator. To define these two operators, let

DOI: 10.14750/ME.2022.020

TABLE 4.3: Sample lattice rules for the artificial word pair
*(xabyx**ab**yz, xabyx**cdw**yz)*

	α^L	σ^L	ω^L	η_f^L	η_b^L	Δ^L			
R_1^L	x	ab	y	2	1	$a \to c$		$b \to d$	$+w$
R_2^L	byx	ab	yz	1	1	$-a$	$+c$	$b \to d$	$+w$

us have two rules:

$$\begin{aligned} R_1^L &= \left(\alpha_1^L,\ \sigma_1^L,\ \omega_1^L,\ \eta_{f1}^L,\ \eta_{b1}^L,\ \langle \delta_{1_i}^L \rangle\right) \\ R_2^L &= \left(\alpha_2^L,\ \sigma_2^L,\ \omega_2^L,\ \eta_{f2}^L,\ \eta_{b2}^L,\ \left\langle \delta_{2_j}^L \right\rangle\right) \end{aligned} \tag{4.6}$$

The intersection of these two rules is a new rule whose components are calculated by intersecting the original components using different methods:

$$\begin{aligned} R_1^L \cap R_2^L = (&\alpha_1^L \cap_{\leftarrow} \alpha_2^L,\ \sigma_1^L \cap_{\leftrightarrow} \sigma_2^L,\ \omega_1^L \cap_{\rightarrow} \omega_2^L, \\ &\eta_{f1}^L \bar{\cap}\, \eta_{f2}^L,\ \eta_{b1}^L \bar{\cap}\, \eta_{b2}^L,\ \langle \delta_{1_i}^L \rangle \cap_{\leftrightarrow} \left\langle \delta_{2_j}^L \right\rangle) \end{aligned} \tag{4.7}$$

The intersection of two characters $c, c' \in \Sigma \cup \{\emptyset\}$ is:

$$c \cap c' = \begin{cases} \emptyset & \text{if } c \neq c' \\ c & \text{otherwise} \end{cases} \tag{4.8}$$

Let $s \in \Sigma^k, s' \in \Sigma^l$ be two strings. The full intersection operation that is used for the core intersection will only return a string if all the characters can be intersected and the two strings are of the same length, otherwise the intersection will be the empty string ϵ:

$$s \cap_{\leftrightarrow} s' = \begin{cases} \epsilon \text{ if } k \neq l \text{ or } \exists i \text{ index } (1 \leq i \leq k) \text{ such that } s_i \cap s'_i = \emptyset \\ s_1 \cap s'_1 + \ldots + s_k \cap s'_k \text{ otherwise} \end{cases} \tag{4.9}$$

For the postfix components, we use a different intersection operator that starts from the left side of the two substrings and intersects each character pair while this character level intersection can be performed:

$$s \cap_{\rightarrow} s' = s_1 \cap s'_1 + \ldots s_m \cap s'_m \tag{4.10}$$

where $m \leq \min\{k, l\}$ and $\forall i$ index $(1 \leq i \leq m)$: $s_i \cap s'_i \neq \emptyset$. Also, $s_{m+1} \cap s'_{m+1} = \emptyset$ or $|s| = m$ or $|s'| = m$.

The prefix components are intersected in the inverse way: we start from the right side of the substrings and intersect each character pair while the aligned characters have an intersection. This is the same as if we reversed the strings, intersected them using $\cap_{\rightarrow}$ and reversed the result back:

$$s \cap_{\leftarrow} s' = \left(s^{-1} \cap_{\rightarrow} s'^{-1}\right)^{-1} \tag{4.11}$$

Position indices are intersected using the $\bar{\cap}$ operator that only produces an output if the appropriate indices of the two input rules are equal, otherwise the output

DOI: 10.14750/ME.2022.020

will be empty. In case of two indices i and i':

$$i \bar{\cap} i' = \begin{cases} i & \text{if } i = i' \\ - & \text{otherwise} \end{cases} \tag{4.12}$$

The transformation lists are intersected similarly to the core strings: if the two transformation lists are equal, then the output rule will have the same list, otherwise the intersection cannot be calculated.

Summarizing the different intersection operators, the intersection of two rules cannot be calculated if any of the following cases apply:

- $\alpha_1^L \cap_{\leftarrow} \alpha_2^L = \epsilon$ and $\sigma_1^L \cap_{\leftrightarrow} \sigma_2^L = \epsilon$ and $\omega_1^L \cap_{\rightarrow} \omega_2^L = \epsilon$
- $\eta_{f1}^L \bar{\cap} \eta_{f2}^L = -$ and $\eta_{b1}^L \bar{\cap} \eta_{b2}^L = -$
- $\left\langle \delta_{1_i}^L \right\rangle \cap_{\leftrightarrow} \left\langle \delta_{2_j}^L \right\rangle = \langle\rangle$

The `IsSubsetOf` operator ($R_1^L \subseteq R_2^L$) plays an important role in determining the parent-child relationships among the rules. This operator can be defined similarly to the intersection operator, but instead of stopping if an intersection does not exist, we return `false`, meaning that $R_1^L \not\subseteq R_2^L$.

Example 4.6 (Intersection, `IsSubsetOf`)**.** *Table 4.4 contains a simple example for the intersection operation. It can also be seen that $R_1^L \cap R_2^L \subset R_1^L$ and $R_1^L \cap R_2^L \subset R_2^L$, as expected.*

TABLE 4.4: Sample rule intersection

	α^L	σ^L	ω^L	η_f^L	η_b^L	Δ^L		
R_1^L	*aei*	*abc*	*dfg*	3	2	$-a$	$b \rightarrow b$	$c \rightarrow l$
R_2^L	*ei*	*abc*	*di*	3	1	$-a$	$b \rightarrow b$	$c \rightarrow l$
$R_1^L \cap R_2^L$	*ei*	*abc*	*d*	3	–	$-a$	$b \rightarrow b$	$c \rightarrow l$

4.1.4 Lattice Builder Algorithms

The training data of the lattice builder algorithm is a set $\mathbb{I} = \{(w_l, w_r)\}$ containing word pairs from the set of meaningful words of the target language $W = \{w\} \subset \Sigma^*$.

Using the Levenshtein distance based rule generation process, we can identify the changing substrings in the base words. From the generated transformation paths, first we remove the invariant replacements from the beginning and the end of the list, because they do not hold additional information.

The responsibility of the lattice builder algorithms is to build a lattice from these generated rules.

Complete Lattice Builder

To build a complete lattice from the generated rules, we need to generate all the intersection rules as well. Then, based on the parent-child relationships determined by the `IsSubsetOf` operator, we insert these rules into an empty lattice.

The structure of the resulting lattice will be similar to the sample lattice in Figure 4.3.

We distinguish the following main node categories:

- Zero (0) node: the infimum of the lattice, containing no transformation rule.

DOI: 10.14750/ME.2022.020

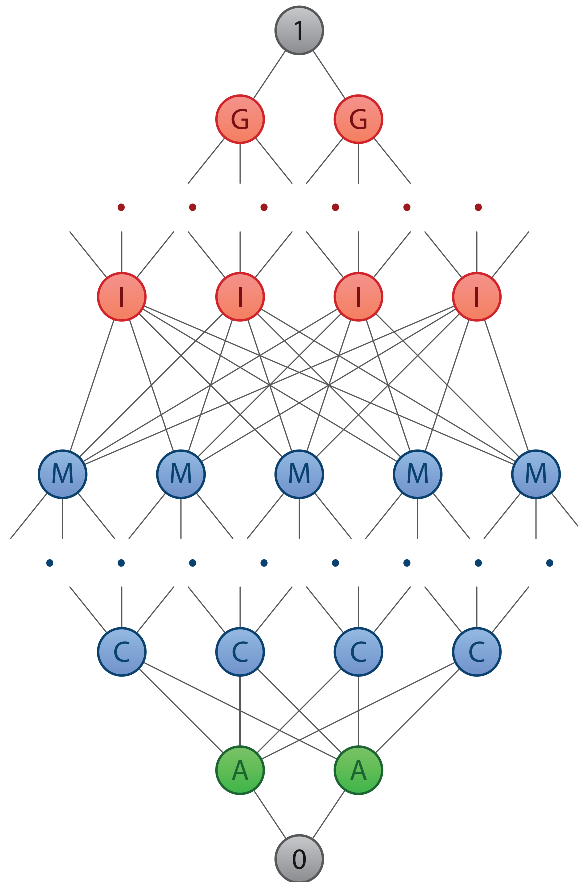


FIGURE 4.3: The structure of a sample lattice

- Atomic (A) nodes: the upper neighbors of the zero node, usually containing the rules generated directly from the training data set.
- Consistent (C) nodes: nodes containing consistent rule intersections. These rules are special because if we apply them on an arbitrary input word, the result will be the same as if we used any of their descendants.
- Maximal (M) consistent nodes: special consistent nodes that only have inconsistent ancestors, i.e. they are the top-level consistent nodes in the lattice.
- Inconsistent (I) nodes: they contain transformation rules whose confidence is low, thus they are usually used only for orientation purposes during node searching.
- General (G) node(s): one or more lower neighbors of the unit node, containing the most general rules that describe the training data set. For example in case of Hungarian accusative case this rule would be something like *"insert a 't' at the end of the word"*.
- Unit (1) node: the supremum of the lattice, containing no transformation rule.

The disadvantage of the complete lattice builder is that the resulting lattice can be very huge because of the large number of intersections.

Consistent Lattice Builder

One way to reduce the lattice size is to eliminate every inconsistent node. Since these nodes contain rules that are usually not used to produce an inflected form (unless they do not have any matching descendants), they can be dropped without losing

DOI: 10.14750/ME.2022.020

much information. This way the maximal consistent nodes of the complete lattice will be the coatoms of the reduced consistent lattice.

There are two cases that the modified algorithm needs to take care of:

1. If a rule intersection is immediately inconsistent, it does not need to be inserted into the lattice.
2. If a previously inserted consistent node becomes inconsistent after inserting new descendants/ancestors, it needs to be removed from the lattice.

Proposition 4.1 summarizes when the consistency of a node may change.

Proposition 4.1. *The consistency of a node can only change if it is initially consistent and a new consistent descendant is inserted into the lattice. In other cases, consistency cannot change.*

Proof. Let n denote an arbitrary node whose consistency is in question. We have to check the following cases:

- If n is inconsistent and we insert a new inconsistent node, consistency cannot change. If the new node is an ancestor of n, it does not influence its consistency, and if it is a descendant or they are not comparable, it does not influence consistency either, as n is already inconsistent and it cannot become consistent.
- If n is inconsistent and the new node is consistent, the same things apply, because the new node will be either not comparable with n or it will be its new descendant.
- If n is consistent and we insert a new inconsistent node, it will definitely be a new ancestor of n if they are comparable, since an inconsistent node is more general than a consistent one, thus consistency of n cannot change.
- If n is consistent and the new node is also consistent, there can be three cases if they are comparable:
  - If the new node is an ancestor, n must remain consistent, since at least one of its ancestors is consistent.
  - If the new node is a descendant and its transformation list can be intersected with the transformation list of n, consistency cannot change.
  - However, if the new node is a descendant and the transformation lists cannot be intersected, it means that there is at least one word pair in the training data set for which the rule of n and the rule of the new node will yield different results, therefore n becomes inconsistent.

□

Minimal Lattice Builder

During inflection generation, the most general consistent rule must be found in the lattice whose context matches the input word. This means that the non-maximal consistent nodes are never accessed. Figure 4.4 demonstrates why: the red node is inconsistent,[2] because there are several different transformations among its descendants; while the yellow node is consistent, since all of its descendants result in the same output.

Therefore, to reduce the lattice size, we can eliminate all the non-maximal consistent nodes. Since the maximal consistent nodes are retained, the inconsistent rules are not dropped either, which means that the appropriate matching rule can be found more quickly than scanning all the maximal consistent nodes linearly.

The minimal lattice builder has two phases:

[2]For inconsistent nodes, the stored list of transformation steps is chosen from the child nodes, using the node with the highest relative frequency.

DOI: 10.14750/ME.2022.020

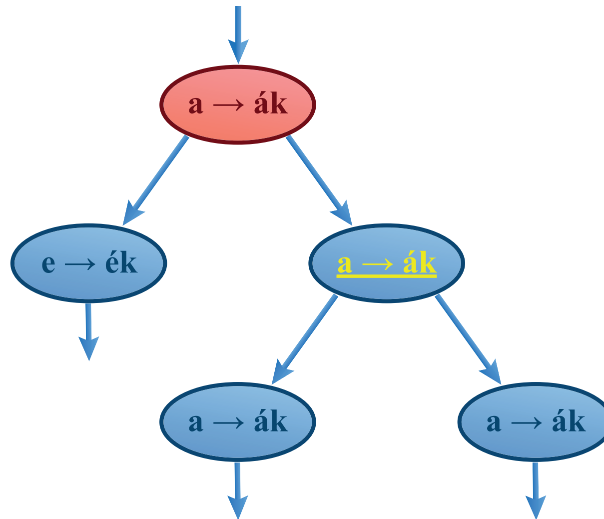


FIGURE 4.4: A sample maximal consistent node

1. It builds a temporary consistent lattice using the consistent lattice builder algorithm.
2. It takes the coatoms (the children of the root) of the temporary lattice and builds a complete lattice from them using the complete lattice builder algorithm.

This way the resulting lattice will only contain the maximal consistent nodes of the original complete lattice, and it will also contain all their intersections.

4.1.5 Inflection Generation

After the lattice is built and an input word is provided whose inflected form should be produced, the model needs to find a matching rule in the lattice whose context is contained by the input word. The search algorithm starts at the unit node and always chooses a child node whose rule still matches the input word. If no matching child rule can be found, or we found a consistent node, the search process can stop. If no consistent matching rule is found, we can still try to apply the lowest matching rule on the input word.

The proposed lattice based model is unfortunately not symmetric, which means that it can be used either during inflection generation or morphological analysis. If we want to use the model during analysis, the training word pairs must be reversed so that the model can learn how to produce the base forms from the inflected forms.

4.2 Atomic String Transformation Rule Assembler (ASTRA)

The Atomic String Transformation Rule Assember (ASTRA) model has a more compact rule model than the one introduced in the previous section. The main characteristics of ASTRA are:

- its atomic rule model that makes ASTRA truly symmetric,
- its simplified set or prefix tree based storage structure, and
- its fast rule selection method during inflection generation and morphological analysis.

DOI: 10.14750/ME.2022.020

4.2.1 The ASTRA Rule Model

Compared to the lattice based model, ASTRA has a more compact rule model that only contains the context of the transformation, the changing substring and the replacement string. In this sense, ASTRA is similar to TASR, but is capable of modeling prefix, postfix and infix transformations as well. Since these rules model simple atomic string transformations in the words, I call them atomic rules.

By relying solely on the context and not position indices, we avoid cases where the rules cannot be applied on the input word due to colliding indices. This way, if the rule context is found in the input word as a substring, the atomic rule is guaranteed to be applicable.

The atomic rules have the following structure:

$$R^A = \left(\alpha^A, \sigma^A, \tau^A, \omega^A\right) \tag{4.13}$$

where

- α^A is the prefix substring, i.e. some characters before the changing substring in the base word form,
- σ^A is the changing substring, i.e. the characters that need to be replaced in the base word form,
- τ^A is the replacement string, i.e. the characters that need to replace σ^A in the base word form, and
- ω^A is the postfix substring, i.e. some characters after the changing substring in the base word form.

During inflection generation, if the model finds an appropriate atomic rule that needs to be applied on the input base word form, the context of the atomic rule must be identified in the word, then σ^A must be replaced with τ^A. The context of the rule is $\gamma\left(R^A\right) = \alpha^A + \sigma^A + \omega^A$.

Example 4.7 (Atomic rules)**.** *Let us have (xabyx**ab**yz, xabyx**cdw**yz) as an artificial word pair. We can create multiple rules that will cover this word pair, two of them can be seen in Table 4.5.*

TABLE 4.5: Sample atomic rules for the artificial word pair *(xabyx**ab**yz, xabyx**cdw**yz)*

	α^A	σ^A	τ^A	ω^A
R_1^A	*x*	*ab*	*cdw*	*y*
R_2^A	*byx*	*ab*	*cdw*	*yz*

As we can see, the rules do not contain any information on position. Also, the transformation is not modeled using elementary steps, but instead a replacement string is provided. Among the two atomic rules in Example 4.7, only R_2^A is unambiguous, since R_1^A matches two positions in the base word form. Having ambiguous rules in the rule base is a problem, so the training algorithm must make sure that the extracted atomic rules are not ambiguous.

4.2.2 The Training Method of ASTRA

Since word-starting and word-ending transformations are very frequent, I introduce two new characters to mark the start ($) and end (#) of the words. These two special characters will help later to determine if the required transformation should occur

DOI: 10.14750/ME.2022.020

at the beginning or at the end of the word. If any of them is part of the context of an atomic rule, the rule cannot be ambiguous. The special characters are not part of the original Σ alphabet. The extended alphabet will be denoted by $\breve{\Sigma} = \Sigma \cup \{\$, \#\}$.

The goal of the training method of ASTRA is to generate and store atomic rules from the $\mathbb{I} = \{(w_l, w_r)\}$ training word pair set. As a first step, the training method will extend all the word pairs with the special characters.

The word extension operator will be denoted by μ: if $w = w_1 \dots w_k$ is a word, then $\mu(w) = \$w_1 \dots w_k\#$. The inverse operator will remove the special character from the input word, i.e. if $\breve{w} = \$w_1 \dots w_k\#$ is an extended word, then $\mu^{-1}(\breve{w}) = w_1 \dots w_k$.

After the extended word pairs are generated, we split them to matching segments:

$$\begin{aligned} \breve{w}_l &= \psi_l^1 \dots \psi_l^k \\ \breve{w}_r &= \psi_r^1 \dots \psi_r^k \end{aligned} \tag{4.14}$$

Here, ψ_l^i and ψ_r^i are substrings of the extended words $\breve{w}_l$ and $\breve{w}_r$. If $\psi_l^i = \psi_r^i$ for an index i, then we call the segment at index i an invariant segment. Otherwise, it is called a variant segment. In a segment decomposition, variant and invariant segments are alternating, and each invariant segment must contain at least 1 character.

There are several valid segment decompositions for each extended word pair, and the training method tries to select the optimal one that correctly isolates the appropriate transformations in the words. The main idea behind this segment decomposition selection is to find the longest aligned invariant segments, and use the remaining variant segments in the decomposition to generate the atomic rules. The fitness function that is used by the training method quantifies the goodness value of the invariant segments:

$$\lambda_1 \cdot \frac{1}{\text{index}_{\text{max}} - \text{index}_{\text{min}}} + \lambda_2 \cdot \left|\psi_l^i\right| \tag{4.15}$$

This formula states that the fitness value is inverse proportionate with the index difference of the two substrings within their words, and proportionate with their lengths. λ_1 and λ_2 are parameters of the training method, while $\text{index}_{\text{max}}$ and $\text{index}_{\text{min}}$ denote the maximum and minimum indices within the words where the segment components appear.

Example 4.8 (Segment decomposition)**.** *One valid segment decomposition for the word pair (dob, ledobott), that means throw and threw down, can be seen in Table 4.6. The middle segment is invariant, while the first and last ones are variant segments.*

TABLE 4.6: Sample segment decomposition for the word pair *(dob, ledobott)*

i	ψ_l^i	ψ_r^i
1	$	*$le*
2	*dob*	*dob*
3	#	*ott#*

For a variant segment $\psi_l^i \rightarrow \psi_r^i$, several atomic rules are generated. The first generated rule whose prefix and postfix components are empty, is called the core

DOI: 10.14750/ME.2022.020

atomic rule. This means that the core atomic rule is a $R_{i0}^A = \left(\alpha_{i0}^A, \sigma_{i0}^A, \tau_{i0}^A, \omega_{i0}^A\right)$ where $\left|\alpha_{i0}^A\right| = \left|\omega_{i0}^A\right| = 0$, $\sigma_{i0}^A = \psi_l^i$ and $\tau_{i0}^A = \psi_r^i$.

The subsequent rules are extended on the left and right sides by one character at a time. Let us assume that $\sum_{j=1}^{i-1} \left|\psi_l^j\right| = n$, $\sum_{j=i+1}^{k} \left|\psi_l^j\right| = m$ and $\left|\psi_l^i\right| = l$. In this case, the extended rules are $R_{ij}^A = \left(\alpha_{ij}^A, \sigma_{ij}^A, \tau_{ij}^A, \omega_{ij}^A\right)$ with the following components ($\forall j, 1 \leq j \leq \min\{n, m\}$):

$$\begin{aligned}
\alpha_{ij}^A &= \breve{w}_l\left[n+1-j,\ n\right] \\
\sigma_{ij}^A &= \psi_l^i \\
\tau_{ij}^A &= \psi_r^i \\
\omega_{ij}^A &= \breve{w}_l\left[n+l+1,\ n+l+j\right]
\end{aligned} \tag{4.16}$$

Here, $w[i, j]$ denotes the substring of w from the ith to the jth character.

However, not all of these atomic rules can be stored in the rule base, because some of them could be ambiguous. Therefore only those rules are retained whose contexts appear only once in the base form of the word. Formally, if there are two indices i_1 and i_2 such that the following conditions apply:

$$\begin{aligned}
&1 \leq i_1 \leq |\breve{w}_l| \\
&1 \leq i_2 \leq |\breve{w}_l| \\
&i_1 \neq i_2 \\
&\breve{w}_l\left[i_1, i_1 + \left|\gamma\left(R_{ij}^A\right)\right|\right] = \gamma\left(R_{ij}^A\right) \\
&\breve{w}_l\left[i_2, i_2 + \left|\gamma\left(R_{ij}^A\right)\right|\right] = \gamma\left(R_{ij}^A\right)
\end{aligned} \tag{4.17}$$

then the extended atomic rule R_{ij}^A is dropped from the rule base due to ambiguity.

Example 4.9 (Atomic rules covering a circumfix)**.** *For the word pair of Example 4.8, the following atomic rules can be generated:*

- *(ϵ, \$, \$le, ϵ)*
- *(ϵ, \$, \$le, d)*
- *(ϵ, \$, \$le, do)*
- *(ϵ, \$, \$le, dob)*
- *(ϵ, \$, \$le, dob#)*
- *(ϵ, #, ott#, ϵ)*
- *(b, #, ott#, ϵ)*
- *(ob, #, ott#, ϵ)*
- *(dob, #, ott#, ϵ)*
- *(\$dob, #, ott#, ϵ)*

The generated and retained atomic rules are stored in so-called rule groups. A rule group is denoted by Γ^A, and it contains all the atomic rules with the same context. This means that $\forall R_i^A, R_j^A \in \Gamma^A$, $\gamma\left(R_i^A\right) = \gamma\left(R_j^A\right)$. The context of the rule group is equal to the context of its atomic rules and is denoted by $\gamma\left(\Gamma^A\right)$.

The ASTRA model can also organize these rule groups in a prefix tree to speed up the search process. If no prefix tree is used, the rules can be processed in parallel, too. These modes will also be examined in Section 4.3.

Example 4.10 (Rule groups)**.** *For the atomic rules in Example 4.9, we can produce 9 different rule groups, each containing a single atomic rule except for the rule group with context \$dob# that contains both (ϵ, \$, \$le, dob#) and (\$dob, #, ott#, ϵ).*

We can easily see that the training of this model is incremental: if a new word pair set is trained, some new atomic rules will be generated, and they will be inserted into existing or new rule groups. Compared to the lattice based model, this requires much less checks and overhead than inserting new nodes into the lattice.

DOI: 10.14750/ME.2022.020

4.2.3 Inflection Generation

The goal during inflection generation is to find some atomic rules in the rule base that match the input word, and apply their transformations on the input. The first step is to extend the input word with the $ and # characters.

In order to find the best atomic rules for the input word, the ASTRA model uses a fitness function that determines how good an atomic rule is for an input extended word:

$$f\left(R^A \mid \breve{w}\right) = \frac{\left|\gamma\left(R^A\right)\right|}{|\breve{w}|} \cdot \theta\left(\gamma\left(R^A\right), \breve{w}\right) \tag{4.18}$$

where the θ function returns how similar the rule context is to the input word. This function can be implemented in different ways, the current implementation simply returns 1 if $\gamma\left(R^A\right) \subseteq \breve{w}$, and 0 otherwise.

Example 4.11 (Fitness value calculation)**.** *Let us have two rule groups, the first containing the atomic rule* $R_1^A = (\epsilon, a\#, i\#, \epsilon)$*, while the other one containing* $R_2^A = (d, a\#, i\#, \epsilon)$*. If the input word is* $\breve{w} = \$ddda\#$*, then*

$$\begin{aligned} f\left(R_1^A \mid \breve{w}\right) &= \frac{|a\#|}{|\$ddda\#|} \cdot 1 = \frac{1}{3} \\ f\left(R_2^A \mid \breve{w}\right) &= \frac{|da\#|}{|\$ddda\#|} \cdot 1 = \frac{1}{2} \end{aligned} \tag{4.19}$$

As $f\left(R_2^A \mid \breve{w}\right)$ *is the larger value,* R_2^A *will be chosen for the transformation.*

Using the fitness function based atomic rule search, we can find the n best atomic rules that match the input word. The next step is to sort them based on their fitness values in a descending order, and start applying the rules on the input word. During inflection generation, this means that we search for $\gamma\left(R^A\right)$ in the base form, then replace σ^A with τ^A.

Sometimes there are several overlapping matching rules that would modify the same part of the word. In these cases, only the first rule is applied from the sorted rule list, since it has the highest fitness value. The other subsequent rules are omitted.

When all the found atomic rules have been applied, we remove the $ and # from the result and return it. Since there might be several valid inflected forms of the same word, we also retry the inflection generation process for a preconfigured times, always dropping the rule with the highest fitness value so that less probable rules can also participate. The model then assigns a weight to the output words. The calculation method of this weight can be implemented in several different ways, for example by returning the average of the fitness values of the applied rules.

Proposition 4.2. *If* (w_l, w_r) *is included in the training word pair set, then for the input word* w_l*,* w_r *will be among the returned inflected forms.*[3]

Proof. During the training phase, the segmentation step will identify the changing substrings, i.e. the variant segments in the word pair. What needs to be shown is that among the generated and retained atomic rules, the one with the highest fitness value will be the one whose context is the word itself.

Based on Equation 4.18, the fitness value will be 0 if the atomic rule context is not contained by the input word. This case can be omitted, since $\gamma\left(R^A\right) = \breve{w}_l$, and the θ function will return 1.

[3] This proposition is also true for morphological analysis.

DOI: 10.14750/ME.2022.020

The left component is $\frac{|R^A|}{|\breve{w}_l|} = \frac{|\breve{w}_l|}{|\breve{w}_l|} = 1$. Since the fitness value must be between 0 and 1, this value is the maximal fitness value. As R^A will be among the matching atomic rules, it will be definitely applied on the input word. □

4.2.4 Morphological Analysis

Since ASTRA is a truly symmetric model, its atomic rules are capable of both generating inflected word forms from base forms and base forms from inflected word forms. The only difference is the way we apply the atomic rules.

During inflection, $\alpha^A + \sigma^A + \omega^A$ must be searched in the input word, then σ^A must be replaced with τ^A. On the other hand, during morphological analysis, $\alpha^A + \tau^A + \omega^A$ must be searched in the input word, and τ^A must be replaced with σ^A.

Of course, the rule groups and the prefix tree (if we use one) are required to be present for both inflection generation and morphological analysis, but the set of rule groups and the prefix tree are also symmetric, meaning that although they have to be present in memory, when we serialize them to disk, we can omit the reversed structures and only store the rule groups and the prefix tree for inflection generation. During loading ASTRA from disk, both directions can be reconstructed from the inflection generation version.

It is also important to note that the atomic rule objects only exist once in memory, and the rule groups only reference them, meaning that having two sets of rule groups does not mean a big overhead, it is only proportionate with the number of rule groups.

4.3 Experiments

In this section I evaluate the lattice based model and ASTRA, including the complete, consistent and minimal lattice builders, as well as the the sequential and parallel ASTRA and ASTRA with prefix tree.

These models are compared with a simple dictionary based system, the FST implementation of Lucene[4] and a custom TASR implementation.[5]

The examined metrics include:

- Average training time: how much time does it take to train the examined models
- Average size: the number of entries in the dictionary, states in FST, nodes in the lattice and rules in ASTRA
- Average search time: how much time does it take to transform an input word using the examined models
- Average accuracy: the percentage of correctly transformed words
  - using pre-trained evaluation word pairs (expected to reach 100%)[6]
  - using disjoint training and evaluation word pair sets

For the performed tests, I select a random training data set containing 10,000 word pairs of Hungarian accusative case,[7] and use 100, 200, . . . , 10,000 word pairs from this set to train the models, gradually extending the size of the training word

[4] `https://lucene.apache.org`

[5] Included in the Morpher framework that can be found on Github: `https://github.com/szgabsz91/morpher`

[6] Normally, morphology models are evaluated using disjoint training and evaluation data sets. I also perform evaluation using this separation, but I also check whether the proposed models can correctly inflect already trained words.

[7] The generation process of the training and evaluation data is described in Section 7.1.

DOI: 10.14750/ME.2022.020

pair set. This means that 100 tests are performed for each model. The number of evaluation word pairs is always 10,000. In case of using pre-trained evaluation word pairs, these 10,000 evaluation word pairs are the same as the randomly selected word pair set, thus the models are expected to eventually reach 100% accuracy. Otherwise, the 10,000 training word pairs and the 10,000 evaluation word pairs are disjoint.

The experiments are repeated 5 times, and the average metric values are calculated. The test machine is a Machbook Pro with 3.1 GHz Intel Core i7 CPU and 16 GB memory in all cases.

4.3.1 Average Training Time

Figure 4.5 displays the average training time of the models in seconds, using logarithmic scale on the y axis. We can see that the training time of the lattice based model is the highest. Among them, the complete lattice builder is the quickest with about 330 seconds after using 10,000 training word pairs. The consistent lattice builder follows with about 406 seconds, because it needs to check node consistency and alter the lattice incrementally during the training phase. Finally, the minimal lattice builder takes about 503 seconds, because of its two-phase nature.

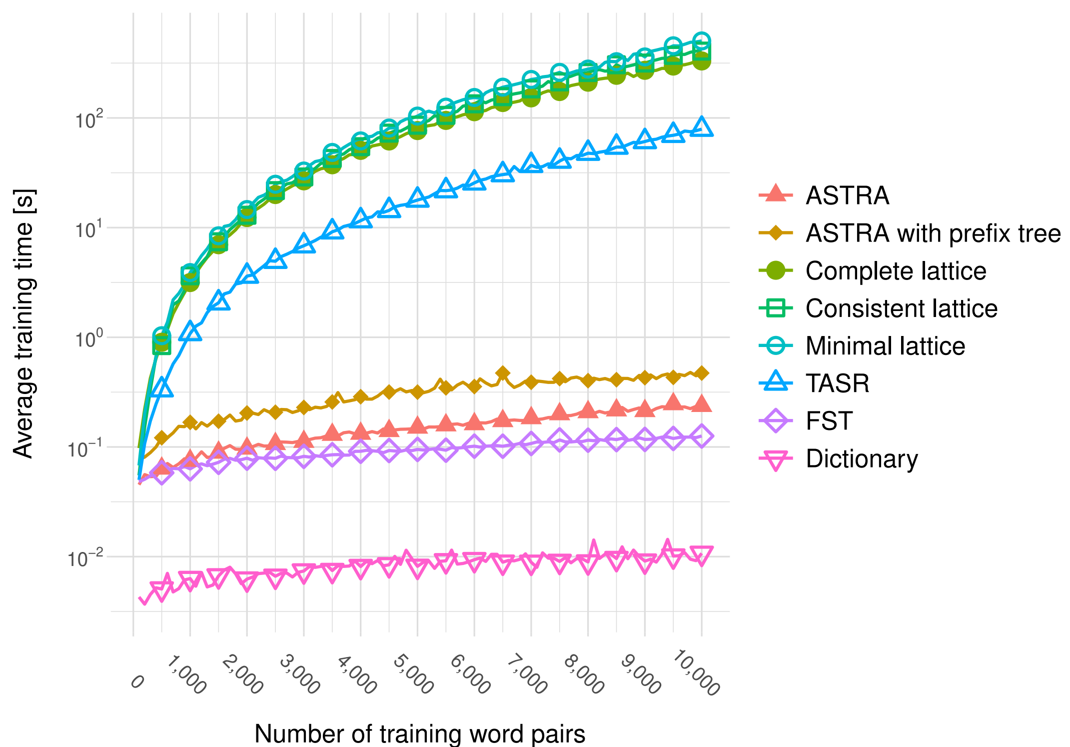


FIGURE 4.5: The average training time of the single-affix transformation engine models

The TASR model has a steep curve, similarly to the lattice model. It reaches 79 seconds using the largest training word pair set. The remaining models have less steep curves. The ASTRA model without the prefix tree can be trained in about 237 ms, while it reaches about 472 ms if we also organize the rule groups in a prefix tree.

The FST implementation of Lucene became the second best model regarding its training time, with about 125 ms, while storing the word pairs in a simple dictionary

DOI: 10.14750/ME.2022.020

takes about 10 ms. However, the last two models have much worse generalization capability, as we will see later.

4.3.2 Average Size

In Figure 4.6 we can see the average size of the built data structures. The dictionary shows a totally linear line, only words with several inflected forms can distort this linearity. Using 10,000 training word pairs, the dictionary had 9,978 entries because of that. Although FSTs act similarly as dictionaries, the FST implementation of Lucene had much more states, namely 20,541.

The TASR model generated several rules for each training word pair, that is why it has an even steeper curve, reaching 55,849 rules when using 10,000 training word pairs. The training method of ASTRA identified even more changing substrings in the base word forms, therefore its size reaches the highest value: 65,894 atomic rules.

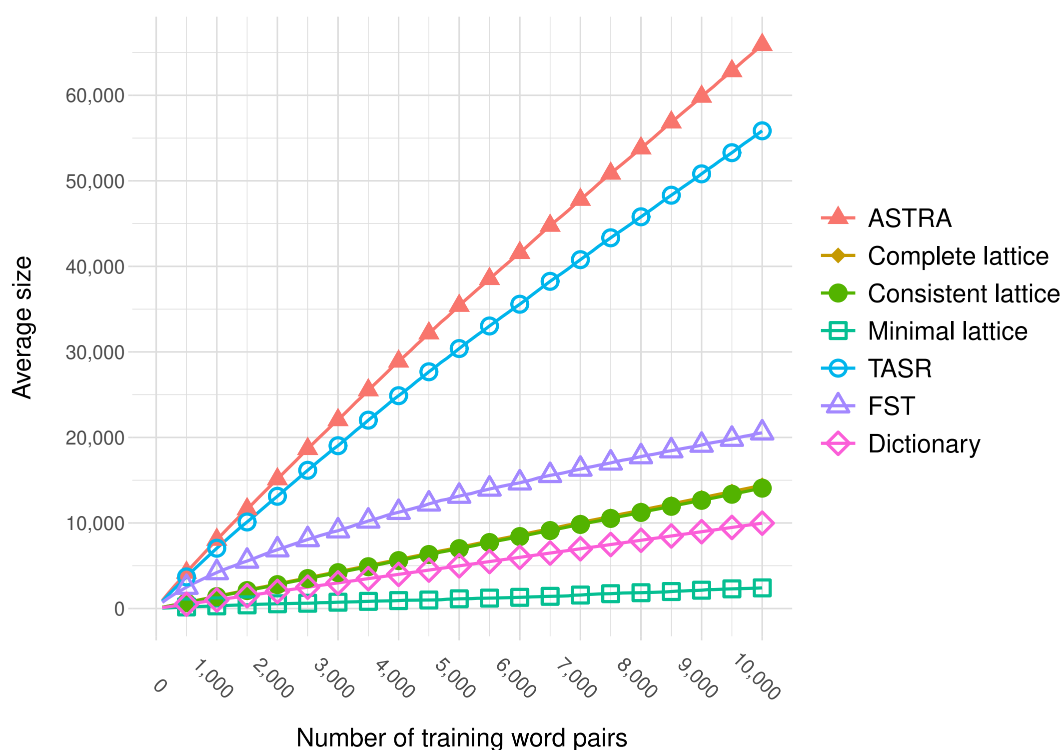


FIGURE 4.6: The average size of the single-affix transformation engine models

As for the lattices, their size was way below the size of the FST model. The complete lattice builder produced 14,404 nodes, while the consistent lattice builder was not far behind with 14,072 nodes. This is interesting because as it turned out, there were not so many inconsistent nodes in the complete lattice, thus eliminating them did not mean a significant size reduction. The minimal lattice builder produced the smallest node set, with only 2,412 nodes.

4.3.3 Average Search Time

Figure 4.7 displays the average search time of the examined models in seconds, using logarithmic scale on the y axis.

DOI: 10.14750/ME.2022.020

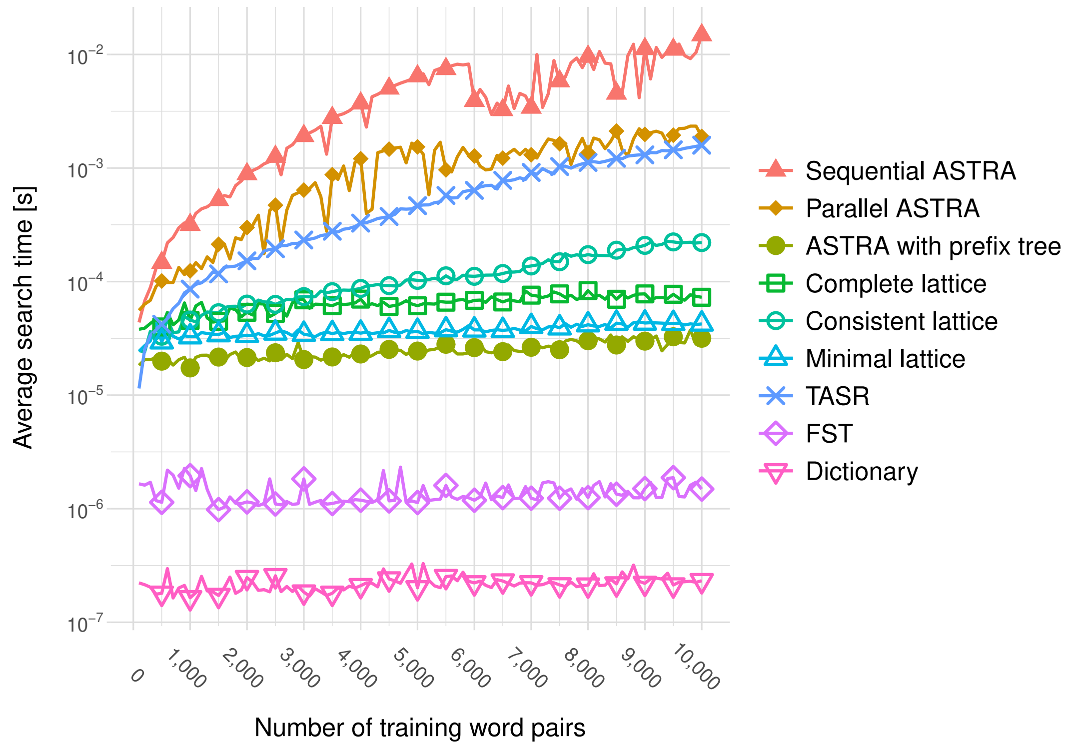


FIGURE 4.7: The average search time of the single-affix transformation engine models

The `java.util.HashMap` based dictionary implementation can return the inflected word forms in nearly constant time, reaching about 232 ns using 10,000 word pairs. The FST implementation of Lucene is similarly quick, with 1.5 μs.

Among the different lattice variants, the consistent lattice produces the worst average search time with about 220 μs, since this model variant uses linear search among the coatoms, as discussed earlier. The complete lattice reaches about 73 μs, however, the minimal lattice produces the best search time with 43 μs.

The slowest model variant is the sequential ASTRA, because the atomic rules are processed sequentially. Using parallel processing, we can improve the 15 ms average response time of the sequential model variant to about 2 ms. This is very similar but slightly better than TASR, the latter one reaching about 1.5 ms.

Using a prefix tree to store the rule groups improves the search time of ASTRA dramatically, reaching about 32 μs, even beating the best lattice based model variant.

4.3.4 Average Accuracy

Figure 4.8 displays the average accuracy of the models when the training data set is the subset of the evaluation data set. Since the last test included the same word pairs for training and evaluation, the accuracy reached almost 100%, as expected. There were a couple of failing words due to some words having multiple inflected forms.

However, we can also see that the generalization capabilities of TASR and especially ASTRA are much better than those of the lattice based model, since they reach 95% much more quickly. The different lattices are very similar, but overall we can say that the minimal lattice is the best among them, then comes the complete lattice and finally the consistent one. The FST and dictionary implementations are not part

DOI: 10.14750/ME.2022.020

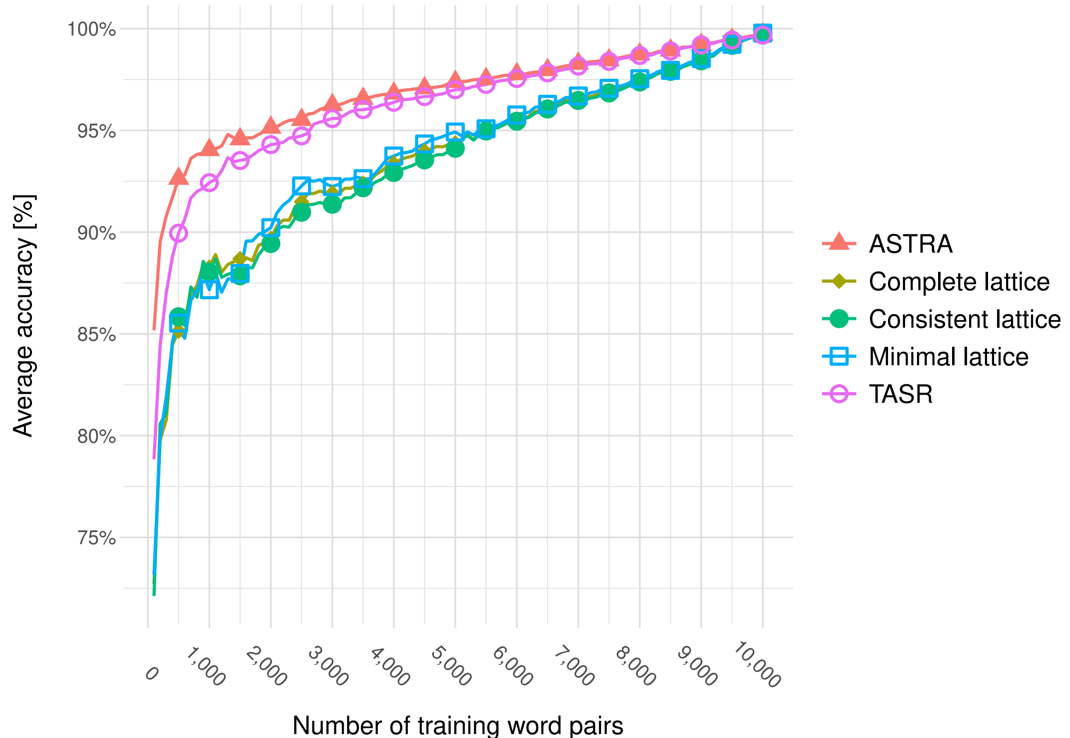


FIGURE 4.8: The average accuracy of the single-affix transformation engine models using subsets of the evaluation word pair set for training

of the figure, but they would show a totally linear line, increasing from 0% to about 100%, since they cannot generalize at all.

Figure 4.9 shows a similar picture: when using disjoint training and evaluation data sets, the ASTRA and TASR models are again on top of the graph. ASTRA reaches 94.06%, while the best result of TASR is 93.79%.

The lines of the three lattices are a bit lower. The accuracy of the minimal lattice is 88.23%, then comes the complete lattice with 88.02%, and finally the consistent lattice reaches 87.57%.

Since the FST and the dictionary can only inflect the pretrained words correctly, they are not displayed on the chart, as they would show a constant 0%.

4.4 Conclusion

In this chapter I proposed two novel single-affix transformation engine models that can learn inflection rules from a provided training word pair set.

The first model (Section 4.1) has a more complex rule structure and stores its rules in a lattice structure. The changing substrings in the words are identified using an improved Levenshtein cost function. I presented three lattice builder methods: the complete, consistent and minimal lattice builder algorithms.

The second model called Atomic String Transformation Rule Assembler or ASTRA (Section 4.2) has a more compact, position independent rule structure that concentrates on simple string transformations. The rules are stored in either a set or a prefix tree. During inflection, the best matching non-overlapping rules are selected using a fitness function, and these rules are applied on the input word one by one.

DOI: 10.14750/ME.2022.020

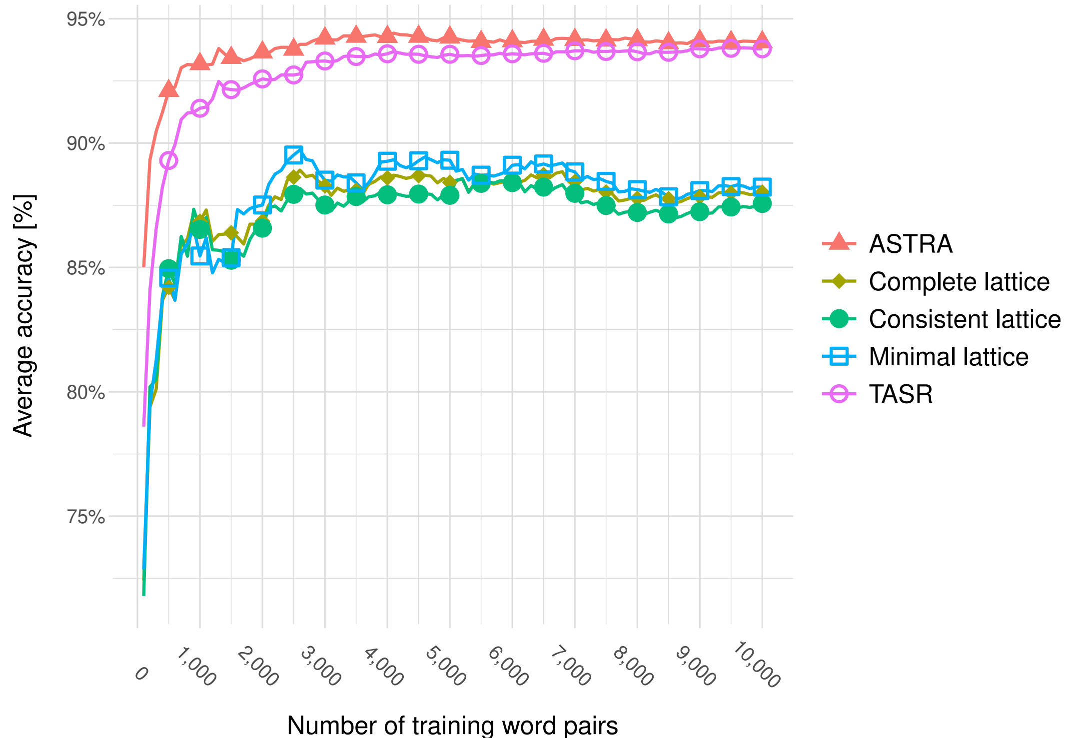


FIGURE 4.9: The average accuracy of the single-affix transformation engine models using disjoint training and evaluation word pair sets

The rule model of this method is symmetric, meaning that the same rules can be used during morphological analysis as well.

Evaluation results (Section 4.3) show that the lattice based model has the most compact rule base, while the ASTRA model has an outstanding average accuracy, training and search time, even though its size is the largest. Only the FST and dictionary implementations beat it regarding training time and search time, but their generalization capability is far worse. The examined metrics are summarized by Table 4.7.

Thesis 2 ***[2] [3] [9] [10] [11] [12] [13] [14] [15] [16]***

I have proposed two novel single-affix transformation engine models that can learn inflection rules from a provided set of training word pairs. The first one is a lattice based model that has a more complex, position dependent rule structure, and stores its rules in a lattice. The second model called ASTRA describes inflection as a set of simple string transformations, omitting the position indices from its rule model. The atomic rules are stored in either a set or a prefix tree based data structure. Both models apply pattern matching during the rule search process. I performed the evaluation of the proposed models, showing that while the lattice based model can achieve minimal storage size, the ASTRA model has an outstanding accuracy for previously unseen words (about 94%), beating the examined baseline models including TASR, FST and a dictionary implementation.

DOI: 10.14750/ME.2022.020

TABLE 4.7: Summary of the measured metrics using 10,000 training word pairs and a disjoint evaluation word pair set

Model	Training time	Size	Search time	Accuracy
Sequential ASTRA	237 ms	65,894	15 ms	94.06%
Parallel ASTRA			2 ms	
ASTRA with prefix tree	472 ms		32 μs	
Complete lattice	330 s	14,404	73 μs	88.03%
Consistent lattice	406 s	14,072	220 μs	87.57%
Minimal lattice	503 s	2,412	43 μs	88.23%
TASR	79 s	55,849	1.5 ms	93.79%
FST	125 ms	20,541	1.5 μs	0%
Dictionary	10 ms	9,978	232 ns	0%

DOI: 10.14750/ME.2022.020

Chapter 5

Multi-Affix Morphology Model

In this chapter I propose a novel multi-affix morphology model called Morpher that can learn the inflection rules of the target language (including all of its affix types), generate inflected word forms from lemmas and affix type sets, and analyze inflected word forms.

In Section 5.1 I introduce the high-level view of the Morpher model, including its main components and their responsibilities.

Section 5.2 contains the necessary formal model to describe concatenative morphology, and it introduces the main features of the Hungarian language that the proposed model needs to take care of.

Section 5.3 is about the training phase of Morpher, i.e. how it learns the inflection rules of all the affix types of the target language, how it calculates the conditional probabilities of affix type chains and how it stores the valid lemmas and parts of speech.

In Section 5.4 I define the necessary operators that make it possible to generate inflected word forms from a given lemma and the set of required affix types.

In Section 5.5, we can read about the operators related to morphological analysis and why this direction is more complex than generating inflected word forms.

Experimental results are summarized in Section 5.6, comparing Morpher with six SIGMORPHON models, three unsupervised segmentation models, and two analyzer models. Results confirm that Morpher has an outstanding average accuracy and generalization capability.

5.1 Architecture of the Proposed Model

The proposed Morpher model has three main components, as demonstrated in Figure 5.1.

- Transformation engines: low-level inflection rule extraction models that are capable of learning the transformations of a single affix type based on a set of training word pairs.
- Probability store: the set of conditional probabilities among all the known affix types of the target language. These probabilities are calculated dynamically during the training phase and updated whenever a new training record is received.
- Lemma store: the store of lemmas and their associated parts of speech.

Morpher coordinates the work of these components to solve the inflection generation and morphological analysis problems in a multi-affix environment in the following way:

- During the training phase (detailed in Section 5.3), Morpher

DOI: 10.14750/ME.2022.020

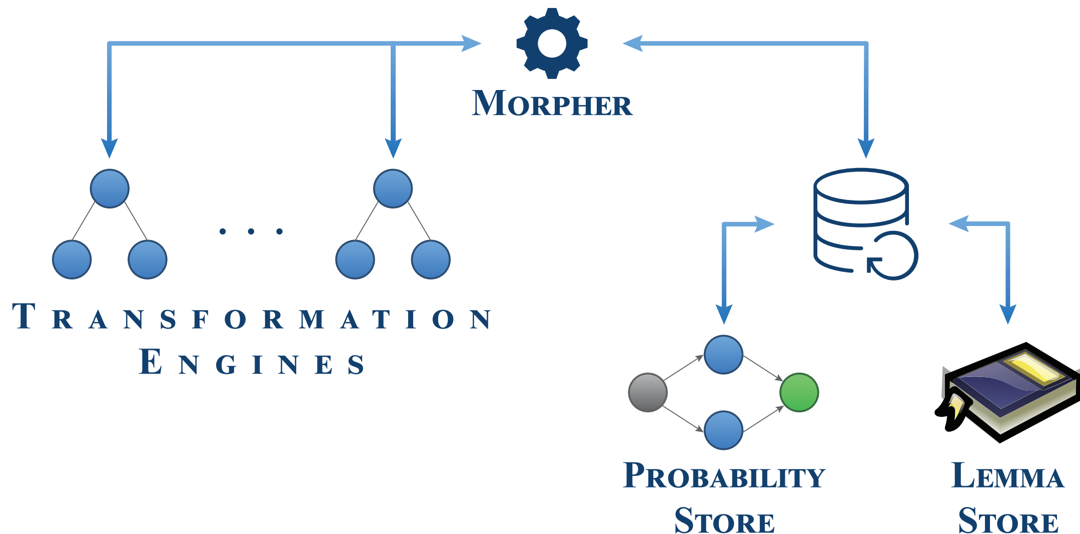


FIGURE 5.1: The main components of the Morpher model

- creates a separate transformation engine instance for each affix type found in the training data set,
- trains the transformation engine instances using dynamically deduced training word pair sets derived from the original training data of Morpher,
- calculates the conditional probabilities of all the valid affix type chains, and
- stores the found lemmas and their associated parts of speech.

- During inflection generation (detailed in Section 5.4), Morpher
  - determines all the valid orders of the given affix types based on the previously calculated conditional probabilities of the affix type chains,
  - has the appropriate transformation engine instances transform the input lemma one by one to produce both the intermediate word forms and the final inflected word form,
  - calculates an aggregated weight for each response it can produce to see how confident the model is in the results, and
  - returns the part(s) of speech, intermediate word form(s) and final inflected form(s), sorting the responses by their aggregated weights in a descending order.
- During morphological analysis (detailed in Section 5.5), Morpher
  - determines which affix types might appear in the received inflected word form,
  - has the appropriate transformation engine instances transform the input inflected form backwards one by one to produce both the intermediate word forms and the final lemma,
  - calculates an aggregated weight for each response it can produce to see how confident the model is in the results, and
  - returns the found affix type(s), intermediate word form(s), part(s) of speech and final lemma(s), sorting the responses by their aggregated weights in a descending order.

As we will see, there are some words in Hungarian that can have several possible lemmas, parts of speech or even inflected forms generated using the same affix type.

DOI: 10.14750/ME.2022.020

To handle these edge cases, Morpher has been designed in a way that it can return multiple responses during both inflection generation and morphological analysis. During evaluation, I will also point out how many responses Morpher produced in average, and what the average index of the correct response was, as these metrics become extremely important when dealing with multiple results.

The key component in learning inflection rules is the set of transformation engines. These engines can be implemented using any morphology model that is capable of learning single-affix transformation rules from a training word pair set. Practically, during the evaluation of Morpher, I will use ASTRA as the transformation engine implementation, as this model performed exceptionally well as demonstrated by the experiments in Section 4.3.

5.2 The Formal Model of Concatenative Morphology

In concatenative morphology, the semantic meaning of words is altered by adding affixes to them. (In Hungarian, this means mainly appending or prepending.) This process is called inflection.

The basis of the target language vocabulary is the set of lemmas, i.e. the grammatically correct root word forms. To distinguish between the set of words (lemmas and inflected forms) and the set of lemmas, words will be denoted by $w \in W$, while lemmas will be denoted by $\bar{w} \in \bar{W} \subset W$. The main difference between a lemma and an inflected word form is that inflected words contain at least one affix.

In simpler cases, the affix is a substring of the word. In more complex cases, however, an affix can also alter the base form of the word. In the Hungarian language we can find several examples for these complex cases. Affixes can be grouped into affix types that determine how the affixes change the meaning of the base form. Affix types will be denoted by $T \in \mathbb{T}$.

Lemmas are associated with one or more parts of speech that indicate their syntactic role in the sentences. The part of speech of a word can be changed by derivational affix types, for instance the word *write* is a verb, while *writing* is a noun. Besides determining the syntactic function of the word, parts of speech also limit the set of affix types that can be applied on the word. Parts of speech will be denoted by $\bar{T} \in \bar{\mathbb{T}}$.

For convenience, let us also define the following projection operators:

- $\lambda : W \to 2^{\bar{W}}$ maps a word to its possible lemmas,
- $\mathfrak{L} : \bar{W} \to 2^{\bar{\mathbb{T}}}$ maps a lemma to its possible parts of speech, while
- $\varphi : W \to \{\langle T_1, \ldots, T_k \rangle\}$ maps a word to its possible affix type lists that can be found in it.

In later sections, I will use these operators when I only care about the possible lemmas of a word form, the possible associated parts of speech of a lemma or the possible affix type list found in an inflected word.

Although it is not frequent in Hungarian, there are cases where a word form has several different lemmas and even several different affix type lists, which means that the proposed model needs to be able to return several responses. Similarly, generating an inflected word form using an affix type might result in several possible responses. Figure 5.2 shows an example of the Hungarian word *oszlat* that is the causative inflected form of the lemmas *oszol* and *oszlik* that both mean *decay*. The word has two possible lemmas and two possible inflected forms based on the subjunctive-imperative affix type: *oszlasson* and *oszlassék*.[1]

[1] Archaic word form denoted by the same morphosyntactic tag

DOI: 10.14750/ME.2022.020

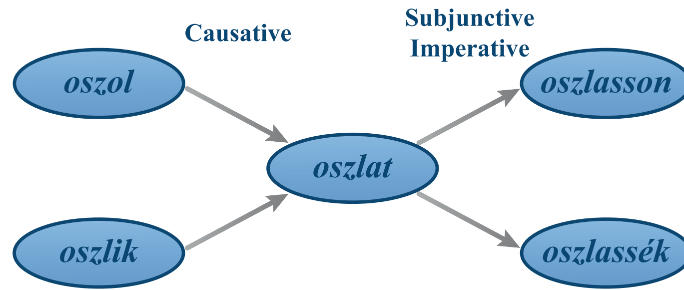


FIGURE 5.2: Multiple lemmas and inflected word forms of the same Hungarian word

Affix types have valid and invalid orders that are defined by the language itself: some affix types can follow each other in a word, others cannot. This adjacency relation will be denoted by $T_i \to T_j$, meaning that T_j can be added to a word whose last affix type is T_i. Parts of speech also participate in this adjacency relation. For example in Hungarian, past tense can only be applied on verbs and not nouns; or accusative case can come after plural, but not vice versa.

Remark 5.1. *In concatenative morphology, every inflected word form is reachable starting from a lemma, applying a number of affix types. This means that the inflected words of a language form a directed graph where only the lemmas have no incoming edges. Formally, $\forall w \in W \setminus \bar{W} : \exists \bar{w} \in \bar{W}$ and $\exists T_1, \ldots, T_k \in \mathbb{T} \ (k \geq 1)$ such that the word w contains the k affix types, $\forall i$ index $(1 \leq i \leq k-1) : T_i \to T_{i+1}$, T_i produces the input word form of T_{i+1} and w is produced by T_k. This relationship is denoted by $\bar{w} \Rightarrow w$.*

Although this is a trivial statement, we can also prove it formally, using the previously introduced notations. Let us assume indirectly that there is at least one word $w_k \in W \setminus \bar{W}$ such that $\nexists \bar{w} \in \bar{W}$ for which $\bar{w} \Rightarrow w_k$. Let us assume that w_k has k affix types $T_1, \ldots T_k \ (k \geq 1)$. Let us apply the inverse transformation of these affix types in reversed order on w_k, producing a word chain consisting of $w_k, w_{k-1}, \ldots, w_1, w_0$, where w_i $(0 \leq i \leq k-1)$ is produced by applying the inverse transformation of the affix type T_{i+1} on w_{i+1}. We can easily see that the last word in this chain is a lemma $\left(w_0 \in \bar{W}\right)$, since it does not have any affix types. This is a contradiction, meaning that the original statement about reachability is true.

Remark 5.2. *In formal language theory, a formal language can be defined as the set of words that can be formed using the rules of a formal grammar. Similarly, the above remark states that all the words of a natural language can be formed using a lemma and a set of affix types.*

In this sense, we can say that the vocabulary of a language forms a directed graph where the nodes are the word forms, the edges represent the applied affix types, and the lemma nodes do not have any incoming edges.

If $\bar{w} \Rightarrow w$, then using the previously introduced projection operators, it is also true that $\bar{w} \in \lambda(w)$.

5.3 The Training Phase of Morpher

The training data of the Morpher model is a $\mathfrak{T} = \left\{\left(w, \bar{w}, \bar{T}, \langle T_i \rangle\right)\right\}$ set, where

- w is an inflected word form,
- $\bar{w}$ is the lemma of w,
- $\bar{T}$ is the part of speech of $\bar{w}$ and

DOI: 10.14750/ME.2022.020

- $\langle T_i \rangle$ is the list of affix types found in w.

During the training phase, Morpher first builds and trains a separate transformation engine instance E_T for each affix type T in the training data $\mathfrak{T}$, to learn its inflection rules.

To learn these rules, the transformation engine E_T needs to receive a set of word pairs (w_l, w_r) demonstrating the characteristics of T. This means that there exists a $\bar{w}$ lemma such that $\bar{w} \in \lambda(w_l) \cap \lambda(w_r)$ and if $\langle T_1, \ldots, T_k \rangle \in \varphi(w_l)$ then $\langle T_1, \ldots, T_k, T \rangle \in \varphi(w_r)$. From such training data, E_T can deduce the appropriate inflection rules.

During inflection generation and morphological analysis, Morpher also needs to know the valid affix type chains of the target language and their conditional probabilities so that it can decide which transformation engine should be used and in what order.

Definition 5.1 (Affix type chain conditional probability). *$\mathfrak{M}$ is a function that can determine the conditional probability of an affix type chain:*

$$\mathfrak{M}\left(\bar{T}_0, T_1, \ldots, T_i\right) = \begin{cases} P\left(\bar{T}_0\right) & \text{if } i = 0 \\ P\left(\bar{T}_0\right) \cdot \prod_{j=1}^{i} P\left(T_j \mid \bar{T}_0, T_1, \ldots, T_{j-1}\right) & \text{if } i = 1, 2, \ldots \end{cases} \tag{5.1}$$

Morpher can calculate the conditional probabilities using relative frequencies:

$$\begin{aligned} P\left(T_n \mid \bar{T}_0, T_1, \ldots, T_{n-1}\right) &= \frac{P\left(\bar{T}_0 \cap T_1 \cap \ldots \cap T_{n-1} \cap T_n\right)}{P\left(\bar{T}_0 \cap T_1 \cap \ldots \cap T_{n-1}\right)} = \\ &= \frac{\left|\left\{w \in W \mid \left\langle \bar{T}_0, T_1, \ldots, T_n \right\rangle \in \varphi(w)\right\}\right|}{\left|\left\{w \in W \mid \left\langle \bar{T}_0, T_1, \ldots, T_{n-1} \right\rangle \in \varphi(w)\right\}\right|} \end{aligned} \tag{5.2}$$

The probability of the part of speech $\bar{T}$ is

$$P\left(\bar{T}\right) = \frac{\left|\left\{w \in W \mid \exists \bar{w} \in \lambda(w),\ \bar{T} \in \mathfrak{L}(\bar{w})\right\}\right|}{|W|} \tag{5.3}$$

Of course in practice, only a subset of W will be available for relative frequency calculation, those words that are part of the training data $\mathfrak{T}$.

If $\mathfrak{M}$ returns 0 for an affix type chain, it means that at least one affix type in the chain cannot come after its predecessors, or formally there is at least one j index $(1 \leq j \leq i)$ such that $T_{j-1} \not\to T_j$.

For morphological analysis, the conditional probabilities of reversed affix type chains (denoted by $\mathfrak{M}^{-1}(T_i, T_{i-1}, \ldots, T_{i-j+1}, T_{i-j})$ where $0 \leq j \leq i$) are calculated similarly, but in reversed order in the affix type chain: starting from the last affix type and moving towards the part of speech.

Finally, the lemmas and their associated parts of speech are also extracted from the training data set. Remembering these items will help Morpher during both inflection generation and morphological analysis:

- During inflection generation, Morpher will know the part(s) of speech of the input lemma, and thus will be able to determine what the first affix type can be.
- During morphological analysis, Morpher will know when it can stop due to having found a grammatically correct root form of the input word, and it will know its associated part(s) of speech.

DOI: 10.14750/ME.2022.020

5.4 Performing Inflection Generation Using Morpher

In this section, I define the inflection generation operator of Morpher. However, since Morpher orchestrates a set of single-affix transformation engine instances, let us first define the forward conversion operator of these lower-level transformation engines.

Definition 5.2 (Forward conversion operator)**.** *The forward conversion operator of the transformation engine instance E_T is a multi-valued function denoted by $\mathcal{FC}^{E_T}$. The range of this operator is a set of word sets, where each set consists of words that have the same lemma as the input word and the same affix type list, except for the additional last affix type (T). Formally, for every $w \in$ domain $\left(\mathcal{FC}^{E_T}\right)$ word in the domain of this operator, $\mathcal{FC}^{E_T}(w) = \{w_1, \dots w_l\}$ such that for all i indices $(1 \le i \le l)$ the following conditions apply:*

- *All of the possible lemmas of the input word are also the lemmas of the output words: $\lambda(w) \subseteq \lambda(w_i)$.*
- *The output words have the same m affix types as the input word, and they append T to the end of the list: $\langle T_1, \dots T_m \rangle \in \varphi(w) \Rightarrow \langle T_1, \dots T_m, T \rangle \in \varphi(w_i)$.*

In the domain of $\mathcal{FC}^{E_T}$, every word w is either a lemma or can be reached from a lemma. Formally, $w \in \bar{W}$ or $\exists \bar{w}_0 \in \bar{W}, \bar{T}_0 \in \bar{\mathbb{T}}$ and $T_1, \dots, T_k \in \mathbb{T}$ such that the following conditions apply:

- *w is reachable from $\bar{w}_0$: $\bar{w}_0 \Rightarrow w$,*
- *$\bar{w}_0 \in \lambda(w)$ is a possible lemma of the w input word,*
- *$\bar{T}_0 \in \mathfrak{L}(\bar{w}_0)$ is a possible part of speech of the $\bar{w}_0$ lemma, $P\left(\bar{T}_0\right) > 0$,*
- *the affix type chain containing $k+2$ elements is valid starting from the part of speech $\bar{T}_0$, i.e. $\mathfrak{M}\left(\bar{T}_0, T_1, \dots, T_k, T\right) > 0$, and*
- *the input word can be found in the output of the kth transformation:*

$$w \in \bigcup_{S \in range\left(\mathcal{FC}^{E_{T_k}}\right)} S \tag{5.4}$$

With this definition in mind, I can now specify the inflection operator of the Morpher model. This operator has two inputs: a lemma and a set of affix types. The goal is to generate the appropriate inflected word form(s) of the input lemma that contain all the given affix types in a valid order. The input does not specify the order of the provided affix types, this must be determined by Morpher. The result is a set of responses, containing

- the part of speech of the given lemma,
- the intermediate word forms with their associated affix types,
- the final inflected word form and
- the aggregated weight of the response that acts as a confidence value.

Definition 5.3 (Inflection operator)**.** *The inflection operator of the Morpher model is a multi-valued function*

$$\mathcal{I} : \bar{W} \times \{T_i\}_{i=1}^m \rightarrow \left\langle \left(\bar{T}_{i_0},\ \left\langle S_{i_j}^{\mathcal{I}} \right\rangle_{j=1}^m,\ \vartheta_i \right) \right\rangle_{i=1}^n \tag{5.5}$$

where

$$S_{i_j}^{\mathcal{I}} = \left(T_{i_j},\ \mathcal{FC}^{E_{T_{i_j}}}\left(w_{i_{j-1}}\right) \right) \tag{5.6}$$

The building blocks of the above definition are the following:

DOI: 10.14750/ME.2022.020

- The input contains the lemma $\bar{w}_0 \in \bar{W}$ and an unordered set of m affix types.
- The output contains a set of n responses where for every i index $(1 \leq i \leq n)$:
  - $\bar{T}_{i_0} \in \mathfrak{L}(\bar{w}_0)$ is a valid part of speech of the input word,
  - $T_{i_1}, \ldots, T_{i_m}$ is a valid permutation of the input affix types $T_1, \ldots, T_m$, i.e. $\mathfrak{M}\left(\bar{T}_{i_0}, T_{i_1}, \ldots, T_{i_m}\right) > 0$,
  - the intermediate and final inflected word forms are generated using the forward conversion operator of the appropriate transformation engine instances, and for every step, the output word form will be the input of the next transformation engine instance, or formally for every j index $(1 \leq j \leq m)$: $w_{i_j} \in \mathcal{FC}^{E_{T_{i_j}}}\left(w_{i_{j-1}}\right)$,
  - each step produces at least one output word, otherwise the whole response is dropped: $\left|\mathcal{FC}^{E_{T_{i_j}}}\left(w_{i_{j-1}}\right)\right| > 0$ for every j index $(1 \leq j \leq m)$,
  - the final output of $\mathcal{I}$ is the inflected form $\mathcal{FC}^{E_{T_{i_m}}}\left(w_{i_{m-1}}\right)$, and
  - the aggregated weights of the given responses are in descending order, i.e. for every i index $(1 \leq i \leq n-1)$, $\vartheta_i \geq \vartheta_{i+1}$.

If the model cannot determine the valid order of the provided affix types, there will be no responses $(n = 0)$. Otherwise, Morpher will provide $n > 0$ responses sorted by their aggregated weights (ϑ_i) in descending order.

The aggregated weight of a response can be calculated in different ways. One possible implementation is to multiply the weights of the steps on the affix type chain. The weight of a single step can be the multiplication of the weight of the output word and the conditional probability of its affix type. Since the conditional probabilities are low due to the large size of the training data set, I scale them into the $[0, 1]$ interval.

Example 5.1 (Inflection generation)**.** *Let us say we would like to inflect the Hungarian word alma (apple) using the accusative case and plural form. Based on an appropriate training set, the engine can determine that the correct order of these affix types is* $\langle$*plural, accusative case*$\rangle$*. The plural form of alma is almák and its accusative case is almákat. The part of speech of alma is noun. If the appropriate probabilities are* $P\,(noun) = \frac{2}{3}$, $P\,(plural \mid noun) = \frac{1}{2}$ *and* $P\,(accusative\ case \mid noun,\ plural) = \frac{1}{2}$*, then*

$$\mathfrak{M}\,(noun,\ plural,\ accusative\ case) = \frac{2}{3} \cdot \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{6}$$

5.5 Performing Morphological Analysis Using Morpher

Similarly to the previous section, let us first define the backward conversion operator of the transformation engine model, since it will be used by Morpher during morphological analysis.

Definition 5.4 (Backward conversion operator)**.** *The backward conversion operator of the transformation engine instance* E_T *is a multi-valued function denoted by* $\mathcal{BC}^{E_T}$*. Every word in the domain of* $\mathcal{BC}^{E_T}$ *can be found somewhere in the range of* $\mathcal{FC}^{E_T}$*, or formally*

$$\forall w \in domain\left(\mathcal{BC}^{E_T}\right) : w \in \bigcup_{S \in range\left(\mathcal{FC}^{E_T}\right)} S \tag{5.7}$$

The output of $\mathcal{BC}^{E_T}$ *for the word* w *is a word set* $\mathcal{BC}^{E_T}(w) = \{w_1, \ldots w_l\}$ *where each output word can be found in the domain of* $\mathcal{FC}^{E_T}$ *such that*

DOI: 10.14750/ME.2022.020

- *all of the possible lemmas of the output words are also possible lemmas of the input word:* $\lambda(w_i) \subseteq \lambda(w)$*, and*
- *the input word has the same* m *affix types as the output words, and it appends* T *to the end of the list:* $\langle T_1, \dots T_m, T \rangle \in \varphi(w) \Rightarrow \langle T_1, \dots T_m \rangle \in \varphi(w_i)$*.*

Using this definition, the morphological analysis operator of the Morpher model can also be specified. This operator only receives an inflected word form as its input. The goal of the model is to identify all the affix types in this word until it finds the lemma(s) and associated part(s) of speech. The result is a set of responses, containing
- the list of affix types found in the input word,
- the intermediate word forms between the lemma and the input word form,
- the lemma,
- the part of speech of the lemma and
- the aggregated weight of the response that acts as a confidence value.

Definition 5.5 (Morphological analysis operator)**.** *The morphological analysis operator of the Morpher model is a multi-valued function*

$$\mathcal{A} : W \to \left\langle \left(\left\langle S_{i_j}^{\mathcal{A}} \right\rangle_{j=m_i}^{1}, \ \bar{T}_{i_0}, \ \vartheta_i \right) \right\rangle_{i=1}^{n} \tag{5.8}$$

where

$$S_{i_j}^{\mathcal{A}} = \left(T_{i_j}, \ \mathcal{BC}^{E_{T_{i_j}}} \left(w_{i_j} \right) \right) \tag{5.9}$$

The following conditions apply on the defined morphological analysis operator:
- The input is an inflected word form $w \in W$.
- The output contains a set of n responses where for every i index $(1 \leq i \leq n)$:
  - $T_{i_{m_i}}, \dots, T_{i_1}, \bar{T}_{i_0}$ is a valid reversed affix type chain, meaning that all conditional probabilities are positive: $\mathfrak{M}^{-1}\left(T_{i_{m_i}}, \dots, T_{i_1}, \bar{T}_{i_0}\right) > 0$,
  - the first word in the chain is the input ($w_{i_{m_i}} = w$) and each consecutive word is an intermediate word form that is generated by applying the backward conversion operator of the appropriate transformation engine instance, i.e. for every j index $(0 \leq j \leq m_i - 1)$: $w_{i_j} \in \mathcal{BC}^{E_{T_{i_{j+1}}}}\left(w_{i_{j+1}}\right)$,
  - each step produces at least one output, otherwise the whole response is dropped: $\left|\mathcal{BC}^{E_{T_{i_j}}}\left(w_{i_j}\right)\right| > 0$ for every j index $(1 \leq j \leq m_i)$,
  - $\bar{w}_{i_0} \in \mathcal{BC}^{E_{T_{i_1}}}\left(w_{i_1}\right) \cap \lambda(w)$ is the lemma of the input word,
  - $\bar{T}_{i_0} \in \mathfrak{L}\left(\bar{w}_{i_0}\right)$ is the part of speech of the lemma, and
  - the aggregated weights of the given responses are in descending order, i.e. for every i index $(1 \leq i \leq n-1)$, $\vartheta_i \geq \vartheta_{i+1}$.

If the model cannot analyze the input word due to not finding a valid lemma or not being able to continue the recursive affix type chain extension, the output will be empty ($n = 0$). Otherwise, Morpher will provide $n > 0$ responses sorted by their aggregated weights (ϑ_i) in descending order.

One big difference that makes morphological analysis more difficult than inflection generation is that we do not know in advance the set of affix types that appear in the input word. Therefore Morpher needs to check much more possibilities during morphological analysis, which means a higher computational complexity. If Morpher already processed the affix types $\left\langle T_{i_{m_i}}, \dots, T_{i_j} \right\rangle$, then the set of affix types that need to be checked as part of the next step is $\left\{ T \mid \mathfrak{M}^{-1}\left(T_{i_{m_i}}, \dots, T_{i_j}, T\right) > 0 \right\}$.

Example 5.2 (Morphological analysis)**.** *Let us say we want to morphologically analyze the Hungarian word rendeltetésem (my function). One possible response that Morpher gives*

DOI: 10.14750/ME.2022.020

us is that the lemma is rendeltetés (function) whose only affix type is the possessor 1st person ⟨POSS ⟨1⟩⟩. However, based on the training data, it can also further expand this lemma: rendel (to order), rendeltet (causative form), rendeltetés (attributive form). Although this is a semantically incorrect chain, the induced rules can yield this response, too, but only with a lower aggregated weight. From this example, we can see that Morpher does not take semantic features into account, so it may provide invalid responses, too. However, these responses are all logical based on the generated rule set, i.e. the training data set.

5.6 Experimental Results

In this section I evaluate the capabilities of the Morpher model, comparing it with state of the art multi-affix morphology models, including six SIGMORPHON models (Helsinki 2016,[2] UF 2017,[3] UTNII 2017,[4] Hamburg 2018,[5] IITBHU 2018[6] and MSU 2018[7]), three unsupervised segmentation models (Morfessor 2.0,[8] MORSEL[9] and MorphoChain[10]), as well as the analyzer models called Lemming[11] and Hunmorph-Ocamorph.[12]

The examined metrics include:

- Average training time: how much time does it take to train the examined models
- Average size: the file size of the exported knowledge base
- Average inflection and analysis time: how much time does it take to inflect or morphologically analyze a word
- Average accuracy: the percentage of correctly inflected and analyzed words
- Average number of responses: how many responses Morpher returns in average
- Average index of the expected response: in average, what is the index of the expected result in the provided list of responses

Some baseline models are not used in all of the evaluation tests. For example although Lemming is a popular morphological analyzer, during its evaluation, I realized that for some reason it does not return the lemma as it should. Therefore this model is not included in the charts, only its average accuracy is mentioned. Hunmorph-Ocamorph is only part of the size comparison, as it had been used to generate the training data. The SIGMORPHON models usually have a CPU and a GPU mode, and since their evaluation was executed only on CPU, they are not included in any execution time comparisons. Finally, MorphoChain is also missing from the comparisons, because it failed with an *OutOfMemoryError* above 50,000 training items.

Besides measuring the above metrics, I also compared the generalization capabilities of the evaluated models, i.e. how well they can handle artificial words that mimic the transformation rules of an existing affix type.

[2] `https://github.com/robertostling/sigmorphon2016-system`
[3] `https://github.com/valdersoul/conll2017`
[4] `https://github.com/hajimes/conll2017-system`
[5] `https://gitlab.com/nats/sigmorphon18`
[6] `https://github.com/abhishek0318/conll-sigmorphon-2018`
[7] `https://github.com/AlexeySorokin/Sigmorphon2018SharedTask`
[8] `https://github.com/aalto-speech/morfessor`
[9] `https://github.com/ConstantineLignos/MORSEL`
[10] `https://github.com/karthikncode/MorphoChain`
[11] `http://cistern.cis.lmu.de/lemming`
[12] `http://mokk.bme.hu/en/resources/hunmorph`

DOI: 10.14750/ME.2022.020

All the tests are performed using a custom pre-generated training and evaluation data set.[13] In the final experiment, I train and evaluate Morpher using the data sets provided by SIGMORPHON.

The evaluation process is similar to that of Section 4.3, only the data volumes are larger: I select 100,000 random training items containing randomly sized affix type chains of every affix type, and use 10,000; 20,000; . . . ; 100,000 records from this set to train the models, gradually extending the training set size. This means that 10 tests are performed for each model. The number of evaluation items is always 10,000; and this set is always disjoint with the training data set. The experiments are repeated 5 times and the average metric values are calculated.

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

5.6.1 Average Training Time

Figure 5.3 displays the average training time in seconds, with logarithmic scale on the y axis. Besides Morpher, only Morfessor 2.0 and MORSEL are included in the chart, since MorphoChain failed above 50,000 training items with an *OutOfMemory-Error* and the SIGMORPHON models were tested only on CPU and not on GPU, so their training took much longer.

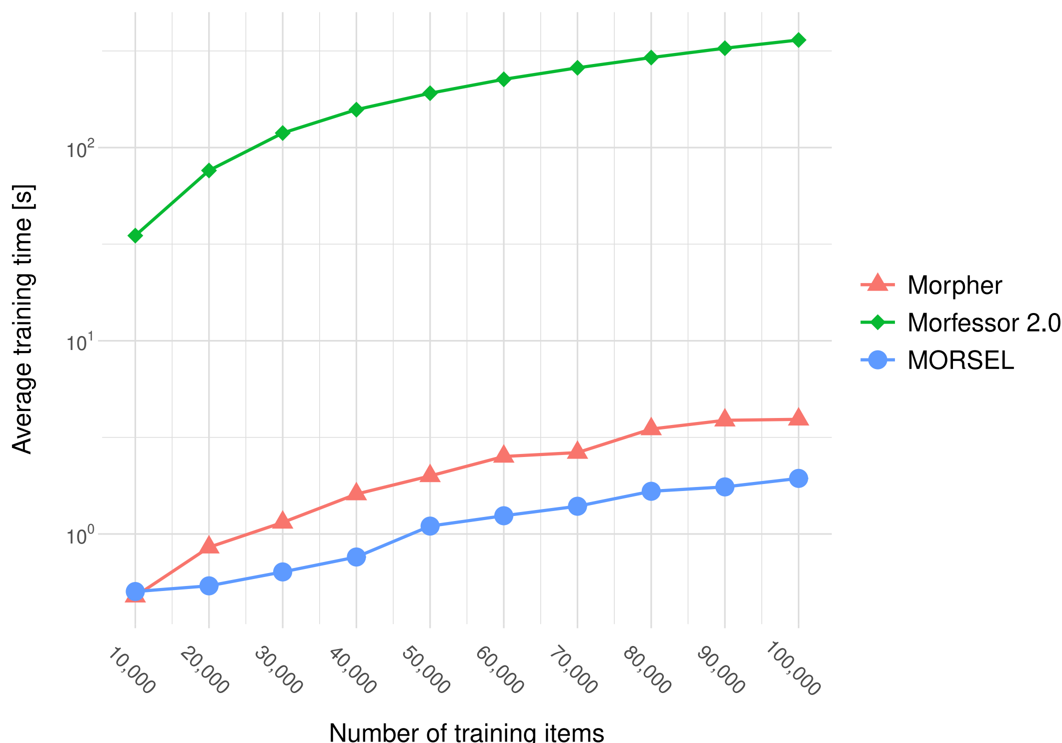


FIGURE 5.3: The average training time of the multi-affix morphology models

Using 100,000 training items, Morfessor 2.0 took 6 minutes to be trained, thus it was the slowest model. The average training time of Morpher was about 3.9 seconds, while MORSEL finished after about 1.9 seconds. However, MORSEL is only a segmentation model, so its knowledge base is probably much simpler.

[13]The data generation process is detailed in Section 7.1.

DOI: 10.14750/ME.2022.020

5.6.2 Average Size

The average size of the models has been measured by exporting their knowledge bases in an external file and checking the size of the exported file. Only Hamburg 2016 and MORSEL do not appear in Table 5.1, because they do not support exporting their knowledge base out of the box.

Morpher is only the 5th model regarding average size after training 100,000 training items, with 8.5 MB. IITBHU 2018 (8.3 MB), UF 2017 (4.5 MB), Morfessor 2.0 (3.5 MB) and MSU 2018 (1.5 MB) have a smaller knowledge base than the proposed model. On the other hand, Hunmorph-Ocamorph that had been used to generate the training and evaluation data sets, has a knowledge base of 22.7 MB. The exported file of Helsinki 2016 is almost 7 times bigger (58.3 MB), and the biggest knowledge base relates to UTNII 2017 with 92.4 MB that is more than 10 times bigger than the file size of Morpher.

For the serialization of Morpher, the protocol buffer data format is used. According to the official Google documentation,[14] *"protocol buffers are a flexible, efficient, automated mechanism for serializing structured data – think XML, but smaller, faster, and simpler"*.

TABLE 5.1: The average file size of the exported knowledge bases

Model	File size
UTNII 2017	92.4 MB
Helsinki 2016	58.3 MB
Hunmorph-Ocamorph	22.7 MB
Morpher	8.5 MB
IITBHU 2018	8.3 MB
UF 2017	4.5 MB
Morfessor 2.0	3.5 MB
MSU 2018	1.5 MB

5.6.3 Average Inflection and Analysis Time

Figure 5.4 displays the average evaluation time of the examined models in milliseconds, with logarithmic scale on the y axis. The chart includes the average inflection and analysis time of Morpher, as well as the average segmentation time of Morfessor 2.0 and MORSEL. MorphoChain is not included because it failed above 50,000 training items with an *OutOfMemoryError*, and the SIGMORPHON models were tested only on CPU and not on GPU, so their average evaluation time was much higher.

As we can see, the average inflection and analysis times of Morpher differ by orders of magnitude, due to the higher computational complexity of the $\mathcal{A}$ operator. While inflection took only about 2.4 milliseconds after training the model with 100,000 training items, morphological analysis took about 2.4 seconds in average.

Although Morfessor 2.0 and MORSEL can only segment the input words, my evaluation shows that they performed segmentation in about 6 minutes and 1.9 seconds, respectively. This means that Morfessor 2.0 became the slowest model during this experiment, while Morpher could analyze the input words in about the same time as MORSEL segmented them, despite morphological analysis being a more complex problem. However, it can be seen that the analysis curve of Morpher is

[14] https://developers.google.com/protocol-buffers/docs/overview

DOI: 10.14750/ME.2022.020

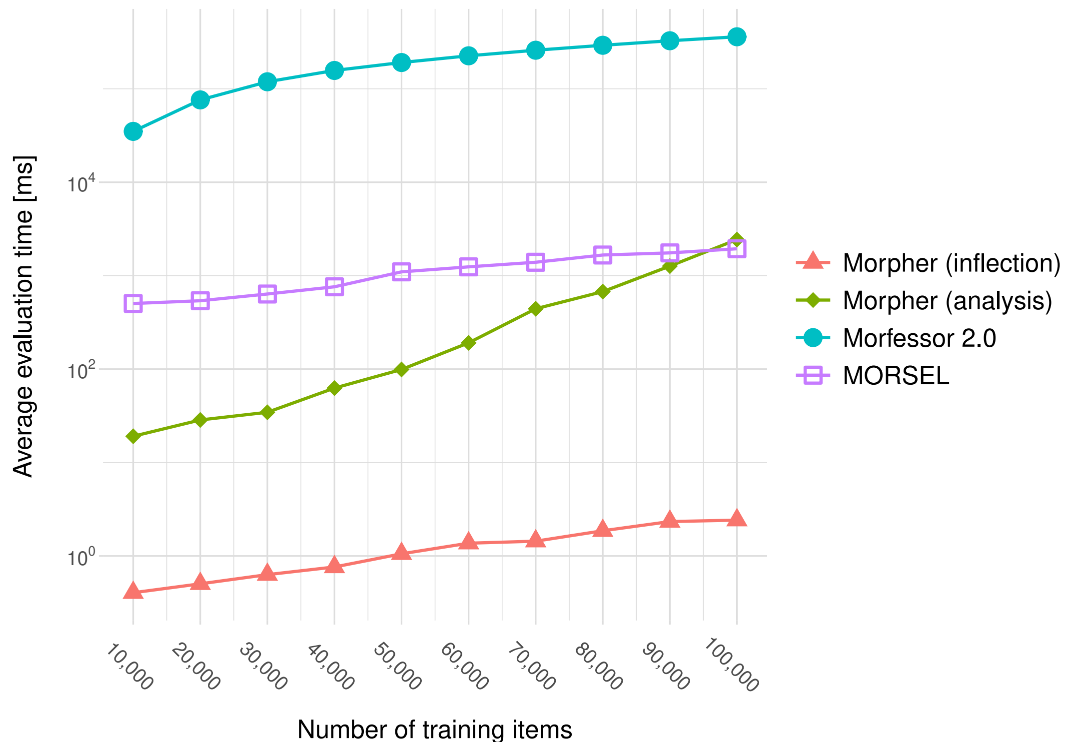


FIGURE 5.4: The average evaluation time of the multi-affix morphology models

steeper, meaning that further extending the training data set would benefit MORSEL in the long run.

5.6.4 Average Accuracy

Figure 5.5 displays the average accuracy of the examined morphology models. As we can see, Morpher has the highest value on the chart, about 97.38%. The closest baseline models include IITBHU 2018 (93.41%) and MSU 2018 (93.26%). The remaining SIGMORPHON models did not reach 90%: Helsinki 2016 achieved 79.23%, UTNII 2017 inflected 88.02% of words correctly, UF 2017 reached 86.32% of average accuracy, and the weakest model became Hamburg 2018 with 69.32%.

As for the segmentation models, they had very poor results. The average accuracy of Morfessor 2.0 became 62.64%, while the result of MORSEL was only 20.11%, even though I only checked if the provided segment lengths matched the correct segmentation of the words, since Hungarian inflection rules often transform even the base form.

I also examined the Lemming morphological analyzer model, however, it had a very low accuracy of about 53%. Moreover, only the affix types were returned by this model, the lemma was not provided, even though Lemming should lemmatize the input words as well.

In case of Morpher, I also measured other metrics besides the average accuracy, since the model can return multiple responses for a single input. The average number of responses means how many responses are given in average for an inflection or analysis task. This metric became about 5.4 for morphological analysis and 37 for

DOI: 10.14750/ME.2022.020

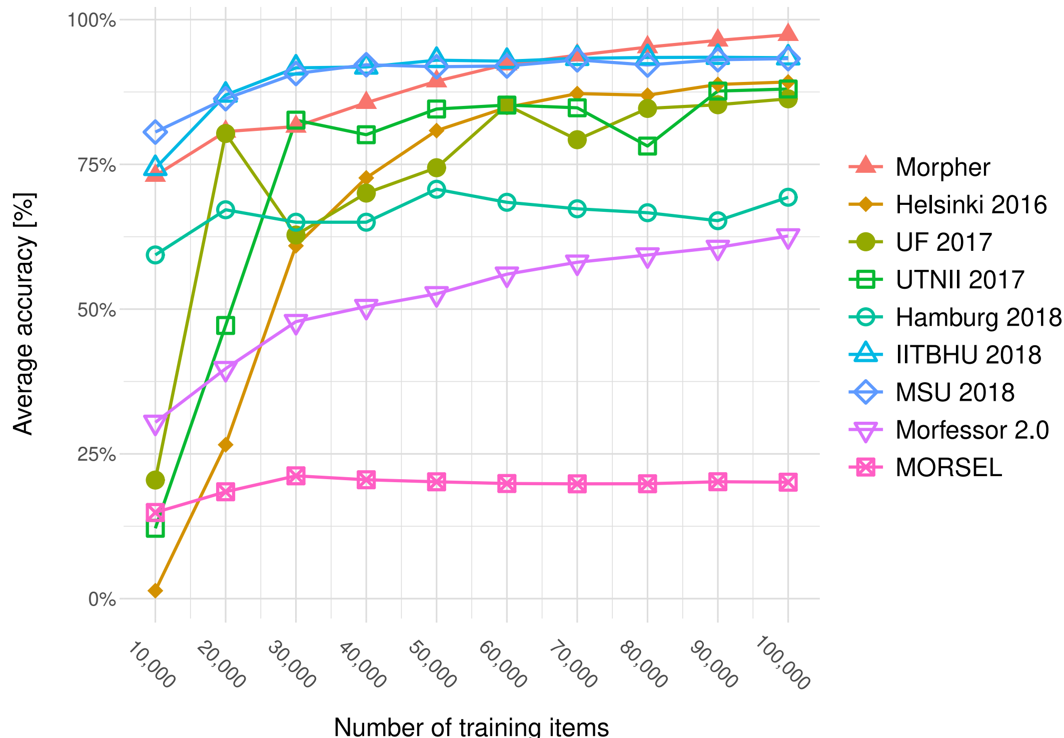


FIGURE 5.5: The average accuracy of the multi-affix morphology models

inflection generation after training the model using 100,000 training items. The difference might be because during morphological analysis, Morpher will omit every response that does not result in a valid lemma, while this restriction is not present for inflection generation.

Since Morpher can provide multiple responses during inflection generation and morphological analysis, it is important to note that if we restrict the number of responses, the accuracy of the model will degrade. In this case, the best-performing SIGMORPHON models will outperform Morpher. I also analyzed the incorrect responses of Morpher, and found that in many cases, those responses that came before the expected one were not incorrect, just less frequent in spoken language, or had a different semantic meaning. It is important to understand that the evaluation data set was selected randomly from an automatically generated data set, which means that there could be other correct responses, too, besides the expected response found in the evaluation data set.

Because of this, another interesting metric is the average index of the expected response, meaning at which index was the expected response provided by Morpher. Optimally, this metric should be 1, i.e. the first response is the expected one, and the others (if any) have a lower aggregated weight. This metric was 1.5 for inflection generation and 2.4 for morphological analysis after training 100,000 items. As we can see, although Morpher gave significantly more results during inflection generation, the expected response almost always had one of the highest aggregated weights. Based on this observation, we could further optimize the model so that it throws away every response with a low aggregated weight. This way the number of responses could be reduced, while retaining the expected (correct) response.

DOI: 10.14750/ME.2022.020

5.6.5 Generalization Capabilities

During evaluation, I also tried to measure the generalization capabilities of the examined models, so that we can see which models can learn the main features of the transformation rules and apply them on artificial words that did not appear in the training data set. For this evaluation, I generated 100 random artificial words containing 3-6 real Hungarian syllables extracted from meaningful words. These syllables were selected and combined randomly. Then, these artificial words were inflected manually, imitating the transformation rules of Hungarian accusative case. These generated words and their inflected forms included items like

- *abajkasztell* → *abajkasztellt*,
- *medarkónunkgótpüf* → *medarkónunkgótpüföt* and
- *öldberczerinc* → *öldberczerincet*.

These artificial word pairs were used to evaluate the generalization capabilities of the examined models. Table 5.2 summarizes the accuracy values.

TABLE 5.2: The accuracy of the multi-affix morphology models using 100 artificial words imitating the inflection rules of Hungarian accusative case

Model	Accuracy
Morpher	95%
Hunmorph-Ocamorph	89%
Morfessor 2.0	80%
Helsinki 2016	41%
UF 2017	0%
UTNII 2017	0%
Hamburg 2018	0%
IITBHU 2018	0%
MSU 2018	0%
MORSEL	0%

As we can see, most of the SIGMORPHON models could not handle these artificial words correctly, except for Helsinki 2016 that achieved only 41% of accuracy. Among the segmentation models, MORSEL could not handle this experiment either, however, Morfessor 2.0 reached 80%. Hunmorph-Ocamorph could analyze 89% of these artificial words correctly, but the best model became Morpher with 95%. This shows that Morpher has an exceptional generalization capability: after training the model with 100,000 real training items, it could analyze even these artificial, meaningless words.

5.6.6 Cross-Validation with the SIGMORPHON Data Sets

According to the comparison of included models published by SIGMORPHON, in 2017 the best models were CLUZH and LMU [Cotterell et al., 2017], reaching about 86%, while the winner in 2018 was the UZH model [Cotterell et al., 2018], achieving about 87% of accuracy. However, their evaluation had been performed using their own data sets, so these results are not comparable with my results on average accuracy.

Therefore I also evaluated Morpher directly using the SIGMORPHON data sets. The main difference between my data sets and those provided by SIGMORPHON is

DOI: 10.14750/ME.2022.020

that since 2017, SIGMORPHON has been publishing three different data sets per language: one normal data set and two others whose data size is significantly smaller. That is a huge challenge for the models, and it is very frequent that a model reaches high percentages for the normal data set and fails for the other two.

TABLE 5.3: The accuracy of the Morpher model using the data sets provided by SIGMORPHON

Data set	Accuracy
SIGMORPHON 2016	98.03%
SIGMORPHON 2017 low	49.64%
SIGMORPHON 2017 medium	54.40%
SIGMORPHON 2017 high	95.14%
SIGMORPHON 2018 low	53.69%
SIGMORPHON 2018 medium	59.92%
SIGMORPHON 2018 high	95.43%

Since my research goal does not include supporting low-resource scenarios, I expected Morpher to perform worse in these cases. As we can see in Table 5.3, Morpher performed exceptionally well for the normal data sets, though: 98.03% for SIGMORPHON 2016, 95.14% for SIGMORPHON 2017 and 95.43% for SIGMORPHON 2018. For the medium-resource and low-resource scenarios, about 50-60% of accuracy was achieved.

5.7 Conclusion

In this chapter I proposed the novel Morpher model to solve the multi-affix inflection generation and morphological analysis problems.

Morpher can be trained using records containing an inflected word form, its lemma, its part of speech and the list of affix types found in the word. From this training data, Morpher can deduce word pairs that demonstrate the transformation rules of each affix type in the target language, calculate the conditional probabilities of all the valid affix type chains and store the lemmas and their parts of speech. To learn the inflection rules of the affix types, Morpher uses a separate ASTRA model instance for each affix type, trained with the appropriate deduced word pairs.

After introducing the main architecture of Morpher, the main features of concatenative morphology and the process of training, inflection generation and morphological analysis in a formal way, I presented the experimental results of the proposed model. Morpher was compared with state of the art morphology models including six SIGMORPHON models (Helsinki 2016, UF 2017, UTNII 2017, Hamburg 2018, IITBHU 2018 and MSU 2018), three unsupervised segmentation models (Morfessor 2.0, MORSEL and MorphoChain), and two analyzer models called Lemming and Hunmorph-Ocamorph.

The measured metrics confirm that Morpher has an exceptional accuracy and generalization capability, and it can be trained and used in acceptable times. The main results using 100,000 training items are summarized by Table 5.4.

The average training time of Morpher was 3.9 seconds, its average inflection time was 2.4 ms, while its average morphological analysis time was 2.4 seconds. Among those baseline models that I could include in the comparison, these metrics are acceptable.

DOI: 10.14750/ME.2022.020

TABLE 5.4: Summary of the measured metrics using 100,000 training items

Model	Size	Accuracy
Morpher	8.5 MB	97.38%
Helsinki 2016	58.3 MB	79.23%
UF 2017	4.5 MB	86.32%
UTNII 2017	92.4 MB	88.02%
Hamburg 2018	–	69.32%
IITBHU 2018	8.3 MB	93.41%
MSU 2018	1.5 MB	93.26%
Morfessor 2.0	3.5 MB	62.64%
MORSEL	–	20.11%
Lemming	–	53%
Hunmorph-Ocamorph	22.7 MB	–

Morpher provided 37 responses for inflection generation and 5.4 responses for morphological analysis in average, but the higher first value can also be reduced by modifying the minimum aggregated weight parameter of the model so that it eliminates all the responses whose weight is lower than this value. The expected response had an average index of 1.5 in case of inflection generation and 2.4 in case of morphological analysis, meaning that the expected correct response almost always had one of the highest aggregated weights in the response list.

I measured the generalization capability in an experiment where I expected the models to be able to correctly handle 100 artificial words, inflected manually imitating the transformation rules of Hungarian accusative case. In this scenario, Morpher achieved an accuracy of 95%, while others could not solve the problem at all, or reached only 89% at most.

I also performed cross-validation using the SIGMORPHON data sets. In the high-resource scenarios, Morpher achieved about 95% that is better than the winner models of 2017 and 2018 that only reached 86-87%.

Thesis 3 ***[3] [4]***

I have proposed a novel multi-affix morphology model called Morpher that can solve the inflection generation and morphological analysis problems, handling all the affix types of the target language. The main feature of the proposed Morpher model is that it builds a separate transformation engine instance for each affix type, and it takes the conditional probabilities of the affix type chains into account during inflection generation and morphological analysis. During the evaluation of Morpher I used the ASTRA model to train the transformation engines. The experiments confirmed the outstanding generalization capabilities and accuracy of Morpher, comparing it with state of the art models including 6 SIGMORPHON models, 3 unsupervised segmentation models and 2 analyzer models.

DOI: 10.14750/ME.2022.020

Chapter 6

Complexity Analysis and Optimization of the Morpher and ASTRA Models

The generalization capabilities and accuracy of the Morpher model seem to be promising, demonstrated by Section 5.6. However, the average inflection time and especially the average analysis time could be reduced to make the model more usable from the viepoint of industrial applications.

In this chapter I perform the space and time complexity analysis of the Morpher and ASTRA models and propose three optimization techniques to reduce their knowledge base and thus make inflection generation and morphological analysis faster. Using optimization, the Morpher model will be able to execute these operations in acceptable, finite time even after training the model with millions of training items.

The space and time complexity analysis is part of Section 6.1, where I analyze the space requirements of the components in Morpher and the time complexity of the steps of its training phase, as well as its inflection generation and morphological analysis operations.

Section 6.2 is about three optimization techniques that can be applied to reduce the rule base of each ASTRA transformation engine instance. The first one eliminates redundant rules, the second one limits the context length of the retained atomic rules, while the third one uses rule attributes such as support and word frequency to drop weaker or less frequent atomic rules. These techniques are controlled using newly introduced model parameters.

In Section 6.3 I empirically analyze the optimization parameters to find the optimal configuration for the experiments of Section 5.6. My goal is to eliminate most atomic rules without degrading the average accuracy.

The evaluation of the optimization techniques is in Section 6.4. First, I re-execute the experiments of the previous chapter using the optimal parameter configuration comparing the new results with those of Section 5.6. Then, I also perform evaluation using 3 million training items to see how well the optimized model can be used with such big training data volumes. Without the newly introduced optimization techniques, the original Morpher model could not handle such data sets at all.

6.1 Complexity Analysis

6.1.1 Space Complexity

Considering the architecture of the proposed Morpher model as described by Section 5.1, the following list highlights the main components that contribute to the

DOI: 10.14750/ME.2022.020

memory and disk space requirements of the model:

- the conditional probabilities of the affix type chains,
- the valid lemmas of the target language and their possible parts of speech and
- the transformation engine instances, including the stored atomic rules in ASTRA.

In the following paragraphs, I analyze the approximate cost of these components.

All the necessary information is extracted from the training data set of Morpher, i.e. $\mathfrak{T} = \left\{ \left(w, \bar{w}_0, \bar{T}_0, \langle T_i \rangle_{i=1}^{k} \right) \right\}$.

Based on this training data set, the number of conditional probability values is equal to the number of valid affix type chains:

$$\Theta \left(\left| \left\{ \left(\bar{T}_0, T_1, \ldots, T_k \right) \mid \mathfrak{M} \left(\bar{T}_0, T_1, \ldots, T_k \right) > 0 \right\} \right| \right) \tag{6.1}$$

This formula describes the values for forwards affix type chains, but similarly, backwards affix type chain conditional probabilities must also be calculated using $\mathfrak{M}^{-1}$.

In the test data[1] used for the evaluation of the proposed models, there are 35,954 different affix type chains. The longest affix type chains contain 9 affix types, while the median of the affix type chain lengths is 5. If we analyzed languages with less complex morphology, these numbers would be much lower.

The size of the lemma database is very language dependent, therefore we cannot formulate exact approximations on it. A rough upper limit of the lemma database size can be approximated with the number of training items at most. In the worst case, every word in the training data set has different lemmas, and as such, the size complexity is $O(|\mathfrak{T}|)$. The number of lemmas in the generated test data set is 121,846. Similarly, the worst case scenario for the associated parts of speech of a single lemma is if the word can be associated with all the possible affix types of the target language: $O\left(\bar{\mathbb{T}}\right)$.

The number of transformation engines is equal to the number of affix types in the target language, so the space complexity here is $\Theta(|\mathbb{T}|)$. In the generated test data set, there are 321 affix types and 12 parts of speech. If the transformation engines are implemented using the ASTRA model, then in each ASTRA instance, there will be a number of stored atomic rules. This metric needs to be understood well, because eliminating unnecessary atomic rules can reduce the average inflection time and the average analysis time significantly.

The number of generated atomic rules depends on the number of deduced word pairs from the original $\mathfrak{T}$ training data set. In the worst case, the number of deduced word pairs is equal to $|\mathfrak{T}|$. This means that we can find a preceding word form for each inflected word in $\mathfrak{T}$. Formally, the number of deduced training word pairs for the transformation engine E_T can be approximated with

$$O \left(\left| \left\{ \left(w, \bar{w}_0, \bar{T}_0, \langle T_i \rangle_{i=1}^{k} \right) \in \mathfrak{T} \mid T_k = T \right\} \right| \right) \tag{6.2}$$

where k is the number of affix types in the word w. Let us denote the set of deduced word pairs demonstrating the transformation rules of the affix type T by $W_{E_T}^2 = \{(w_{il}, w_{ir})\}$. Equation 6.2 approximates the upper size limit of this word pair set.

For every deduced word pair, we generate several atomic rules. Let us examine the approximation of the number of generated atomic rules for the word pair (w_l, w_r). This approximation is $\max(|w_l|, |w_r|) - |\sigma^A|$, where σ^A is the changing

[1] Section 7.1 describes the data generation process in details.

DOI: 10.14750/ME.2022.020

substring component of the generated atomic rules. For the whole transformation engine E_T, the approximation of the generated atomic rules is

$$O\left(\left|W_{E_T}^2\right| \cdot \max_j \left(\max\left(|w_{jl}|, |w_{jr}|\right) - \left|\sigma_j^A\right|\right)\right) \tag{6.3}$$

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

6.1.2 Time Complexity

The training phase of the Morpher model consists of three main parts:

- steps with $O(1)$ time complexity like storing the lemmas and their possible parts of speech,
- deducing training word pairs from $\mathfrak{T}$ for each affix type and
- generating atomic rules from the deduced training word pairs for each affix type.

Deducing training word pairs from $\mathfrak{T}$ generally has a $O\left(|\mathfrak{T}|^2\right)$ time complexity, because every possible item pair must be found that contain adjacent word forms based on their affix type chains. However, this can be optimized by not checking all the other items for each training item, only those that have the same lemma. This will result in the same deduced word pair set, since no word pairs can be generated from training item pairs having different lemmas. Thus, the approximation of deducing training word pairs for the lemma $\bar{w}$ can be reduced to $O\left(\left|\left\{\left(w, \bar{w}_0, \bar{T}_0, \langle T_i\rangle_{i=1}^k\right) \in \mathfrak{T} \mid \bar{w}_0 = \bar{w}\right\}\right|\right)$.

For those training items that have only one affix type, the appropriate training word pair can be deduced in $O(1)$ time, since it contains the base form ($\bar{w}_0$) and the inflected form (w), and thus there is no need to search additional training items.

For every training word pair, we first have to find the variant segment, i.e. the changing substring component of the core atomic rule. For the word pair (w_l, w_r) this can be done in $O(\max(|w_l|, |w_r|))$ time. Generating all the necessary atomic rules just means to extend the context with one character from the left and right sides at a time. This means that the generation time can be approximated with the number of generated atomic rules (see Equation 6.3). Based on these observations, the whole generation process can be done in $O\left(\max(|w_{jl}|, |w_{jr}|) - \left|\sigma_j^A\right|\right)$ time per word pair.

Let us analyze the time complexity of inflection generation. The first task to be solved is to generate all the valid affix type chains containing the input k affix types. In worst case, this can be done in at most $O(k!)$ steps, if all the possible affix type permutations are valid. For every possible permutation, we need to go through the affix type chain and perform the transformation operation of the appropriate transformation engine. 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 the number of atomic rules: $O\left(\left|\left\{R^A\right\}\right|\right)$. This will be the same as Equation 6.3.

As for morphological analysis, the provided input does not contain any information about the number of affix types found in the input word. Morpher will not know which affix types to test either. In the worst case, the input word will contain $O(|\mathbb{T}|)$ affix types, but in practice the concrete number will be significantly lower. At every step, the number of atomic rules to process and potentially apply can be approximated with Equation 6.3. The average analysis time can get high quickly,

DOI: 10.14750/ME.2022.020

because after every processed affix type, Morpher needs to test all those affix types that can appear before the lastly processed one. In worst case, the number of such affix types is $|\mathbb{T}| - 1$, i.e. all the remaining affix types. This means that morphological analysis can be performed in exponential time, roughly in $O\left(|\mathfrak{T}|^{|\mathfrak{T}|}\right)$ time at most.

6.2 Optimization Techniques

In this section, I propose new optimization techniques to reduce the number of retained atomic rules during the training phase of the ASTRA instances. Although there are other possible ways of optimization, this way I expect to make inflection generation and morphological analysis significantly faster. However, it is also important to drop rules in such a way that we keep the average accuracy high. Another goal is to keep the average number of responses and the average index of the expected response relatively low.

6.2.1 Eliminating the Redundant Atomic Rules

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

Definition 6.1 (Redundant atomic rule)**.** *The atomic rule* $R_i^A = \left(\alpha_i^A, \sigma_i^A, \tau_i^A, \omega_i^A\right)$ *is a redundant rule if and only if there exists another* $R_j^A = \left(\alpha_j^A, \sigma_j^A, \tau_j^A, \omega_j^A\right)$ *atomic rule in the rule base such that* $\gamma\left(R_j^A\right) \subseteq \gamma\left(R_i^A\right)$, $\sigma_i^A = \sigma_j^A$ *and* $\tau_i^A = \tau_j^A$. *In this case we say* R_i^A *is covered by* R_j^A.

Example 6.1 (Redundant atomic rule)**.** *Let us take a look at two rules:* R_1^A *= (alm, a, át, #) and* R_2^A *= (*ϵ*, a, át, #). The contexts of these rules are* $\gamma\left(R_1^A\right)$ *= alma# and* $\gamma\left(R_2^A\right)$ *= a#, respectively. If both rules are in the rule base, we can say that* R_1^A *is redundant, since* $\gamma\left(R_2^A\right) \subseteq \gamma\left(R_1^A\right)$ *and their transformations are also the same (*$\sigma_1^A = \sigma_2^A$ *and* $\tau_1^A = \tau_2^A$*), i.e.* R_1^A *is covered by* R_2^A.

Example 6.2 (Non-redundant atomic rule)**.** *If* R_3^A *= (toll,* ϵ*, at, #) and* R_4^A *= (l,* ϵ*, t, #) are part of the same rule base, they do not cover each other. Although l# =* $\gamma\left(R_4^A\right) \subseteq \gamma\left(R_3^A\right)$ *= toll#,* R_3^A *is not redundant, because the transformations are different:* τ_3^A *= at, while* τ_4^A *= t.*

The elimination of redundant rules can be performed during the training phase of each ASTRA instance, there is no need for additional operations. For the word pair (w_{il}, w_{ir}), ASTRA would generate the atomic rules $R_{i0}^A, R_{i1}^A, \ldots, R_{ik_i}^A$, where R_{i0}^A is the core atomic rule with minimal context, while the others are the extended atomic rules. The atomic rules with shorter contexts are more general, while rules with longer contexts are more specific.

In order to reduce the number of generated atomic rules per word pair (k_i at most), I introduce a new parameter p_{max} that identifies the maximum number of atomic rules to generate for each word pair. Using this new optimization parameter, ASTRA will only store the atomic rules $R_{i0}^A, R_{i1}^A, \ldots, R_{il_i}^A$ for the word pair (w_l, w_r), where $l_i = \min\left(p_{max}, k_i\right)$. Those atomic rules that have a longer context than $R_{il_i}^A$ are simply omitted.

Proposition 6.1. *Using* $p_{max} = 1$*, storing only* R_{i0}^A *for each* (w_{il}, w_{ir}) *training word pair and dropping the other* $R_{i1}^A, \ldots, R_{ik_i}^A$ *results in the same rule base as if we generated every possible atomic rule and then dropped all the redundant atomic rules.*

DOI: 10.14750/ME.2022.020

Proof. According to Definition 6.1, an atomic rule $R_i^A = (\alpha_i^A, \sigma_i^A, \tau_i^A, \omega_i^A)$ is redundant if and only if there exists another atomic rule $R_j^A = \left(\alpha_j^A, \sigma_j^A, \tau_j^A, \omega_j^A\right)$ such that $\gamma\left(R_j^A\right) \subseteq \gamma\left(R_i^A\right)$, $\sigma_i^A = \sigma_j^A$ and $\tau_i^A = \tau_j^A$.

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

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

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

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

□

Using the p_{max} optimization parameter, the space and time complexity of Morpher gets simplified. For the transformation engine instance E_T, the number of generated atomic rules becomes

$$O\left(\left|W_{E_T}^2\right| \cdot \min\left(p_{max}, \max_j\left(\max\left(\left|w_{jl}\right|, \left|w_{jr}\right|\right) - \left|\sigma_j^A\right|\right)\right)\right) \tag{6.4}$$

which means, no matter how many atomic rules could be generated per word pair, the model only generates p_{max} at most.

If $p_{max} = 1$, this formula gets simplified even more: $O\left(\left|W_{E_T}^2\right|\right)$.

6.2.2 Limiting the Generalization Factor

The potential problem with p_{max} optimization, especially using $p_{max} = 1$ is that the Morpher model may overgeneralize due to only retaining atomic rules with very short contexts. This means that during inflection generation and morphological analysis, the ASTRA instances will not be able to distinguish among the possible output word forms, their confidence values will be the same for all responses. Therefore, the number of responses will increase and the index of expected response will become random.

In order to avoid this overgeneralization effect, I introduce another optimization parameter called p_{gen}. This parameter identifies the minimum context length of the generated atomic rules, i.e. for every retained atomic rule, $\left|\gamma\left(R^A\right)\right| \geq p_{gen}$. Those atomic rules that do not match this condition are omitted during the training phase and not stored in the rule base.

The p_{gen} and p_{max} optimization parameters can be used together: p_{max} limits the generated and retained set of atomic rules from above, omitting the rules with longer contexts, while p_{gen} does the same thing from below, omitting the rules with shorter contexts. By applying both of them, we can retain a slice of all the possible atomic rules per word pair. For example, $p_{max} = 2$ and $p_{gen} = 3$ will retain rules whose context contains at least three characters, but only two of these rules per each word pair.

DOI: 10.14750/ME.2022.020

This way, we drop the most general rules ($\left|\gamma\left(R^A\right)\right| = 1$ and $\left|\gamma\left(R^A\right)\right| = 2$), as well as the most specific rules where $\left|\gamma\left(R^A\right)\right| \geq 2 + 3 = 5$ in case of suffix rules. Choosing the right combination of p_{max} and p_{gen} is important so that the model can eliminate the most atomic rules while maintaining a relatively high average accuracy.

Using the p_{gen} optimization parameter, the space and time complexity of Morpher gets simplified, similarly to p_{max}. However, the modified formula is not as clear as in the previous subsection. The number of generated atomic rules for the transformation engine instance E_T can be approximated with:

$$O\left(\left|W_{E_T}^2\right| \cdot \max_j \left(\max\left(\left|w_{jl}\right|, \left|w_{jr}\right|\right) - \left|\sigma_j^A\right|\right) - \Upsilon_{p_{gen}}\right) \tag{6.5}$$

where $\Upsilon_{p_{gen}}$ denotes the minimum number of generated atomic rules that have a context shorter than p_{gen}.

Asymptotically, this means that in the worst case, no reduction occurs, if all the generated atomic rules have a context at least as long as p_{gen}.

6.2.3 Indirect Noise Reduction

Unlike the p_{max} and p_{gen} optimization parameters, we can also use global information for atomic rule reduction, thus performing an indirect noise reduction: those atomic rules that are not strong enough based on some metrics, are dropped from the rule base.

One such metric is the so-called support value, that means how many word pairs there are in the set of deduced training word pairs that the generated atomic rule matches. Those atomic rules that are generated from more word pairs become stronger. When the training algorithm finds that a generated atomic rule is already stored in the rule base, it increases its support value.

Another metric is the word frequency, that also contains information about the data source of $\mathfrak{T}$. In Section 7.1 we can read about the training data generation process. The data source of this data generation was documents on the internet, containing free texts. The frequency of a word is the number of its occurrences in these free texts. The word frequency of an atomic rule is the sum of frequencies of the words it matches.

Based on these metrics, I introduce two new optimization parameters: p_{supp} and p_{freq}. Using these optimization parameters, every atomic rule will be dropped from the rule base whose support or word frequency is lower than p_{supp} and p_{freq}, respectively. In Section 6.3 we will see which one of these parameters achieve better size reduction.

The space and time complexity of Morpher will be simplified using these optimization parameters, but the scale of this simplification cannot be approximated formally, since it highly depends on the training data quality.

6.3 Empirical Analysis of the Optimization Parameters

In this section, I empirically analyze the three optimization techniques using 100,000 random training items to decide how much rule base elimination can be achieved using them, and how some of the previously measured metrics such as the average accuracy, the average number of responses and the average index of the expected response change based on the number of retained atomic rules.

DOI: 10.14750/ME.2022.020

First, let us analyze p_{supp} and p_{gen}. In Figure 6.1 we can see the average accuracy based on the number of retained atomic rules, with logarithmic scale on the x axis. On the right side of the diagram, no reduction took place and the total number of generated atomic rules was 578,497. As we move towards the left side, the number of retained atomic rules decreases. The chart contains three lines.

The lowest line is for random elimination, where I drop the atomic rules randomly from the generated rule base. We can see that the average accuracy is reduced dramatically, because no rule properties are taken into account while choosing the rules to be eliminated.

The other two lines are for the support and word frequency based optimization techniques. These two lines are very close to each other, and as I start to eliminate the atomic rules with the lowest support values and word frequencies, the average accuracy does not start to significantly reduce until the number of retained atomic rules drops below about 10,000 (marked with a vertical line). This means that even for about 1.73% of the original rule base, the average accuracy is still around 93%.

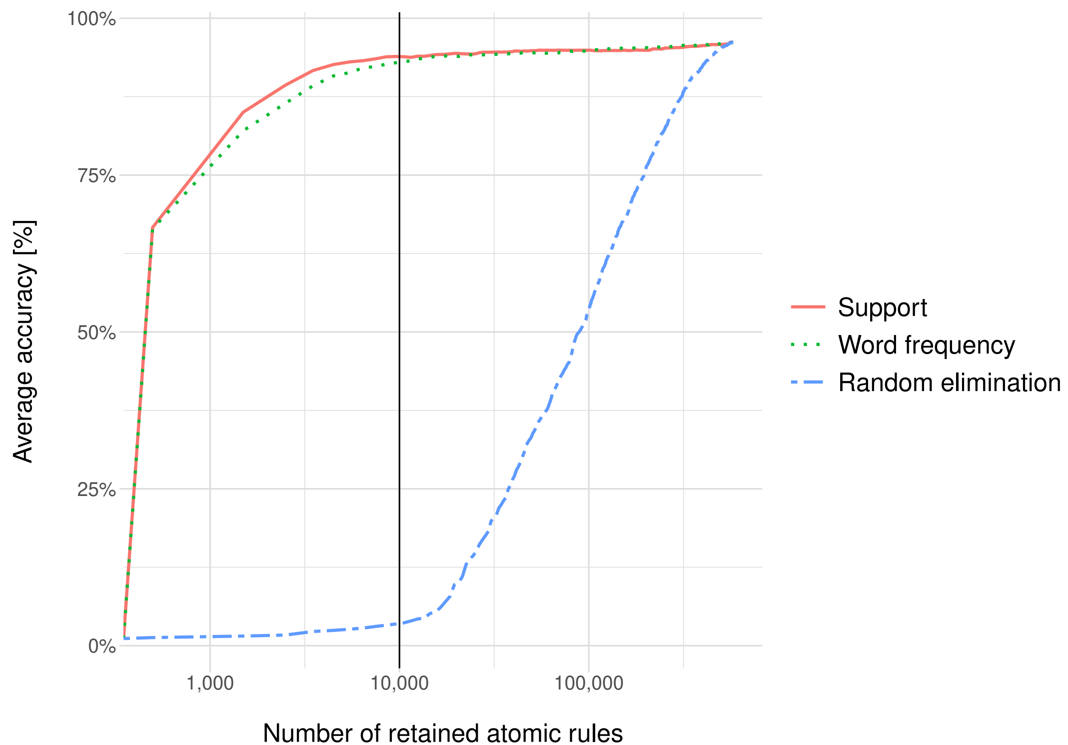
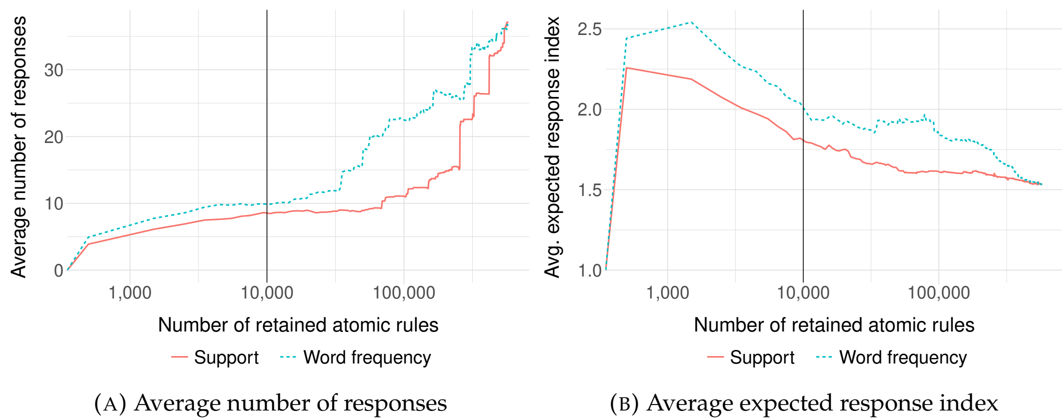


FIGURE 6.1: The average accuracy based on the number of retained atomic rules using p_{supp} and p_{freq} optimization

We can also see from the figure that the support value based optimization provides a slightly higher average accuracy than the word frequency based optimization.

In Figure 6.2 we can see the average number of responses (left side) and the average index of the expected response (right side). Based on these charts, we can see that eliminating the unnecessary atomic rules has a positive impact on these metrics. Retaining 10,000 atomic rules, the average number of responses during inflection decreases from 37 to about 8.4 in case of the support value optimization and 8.6 in case of the word frequency based optimization. However, the average

DOI: 10.14750/ME.2022.020

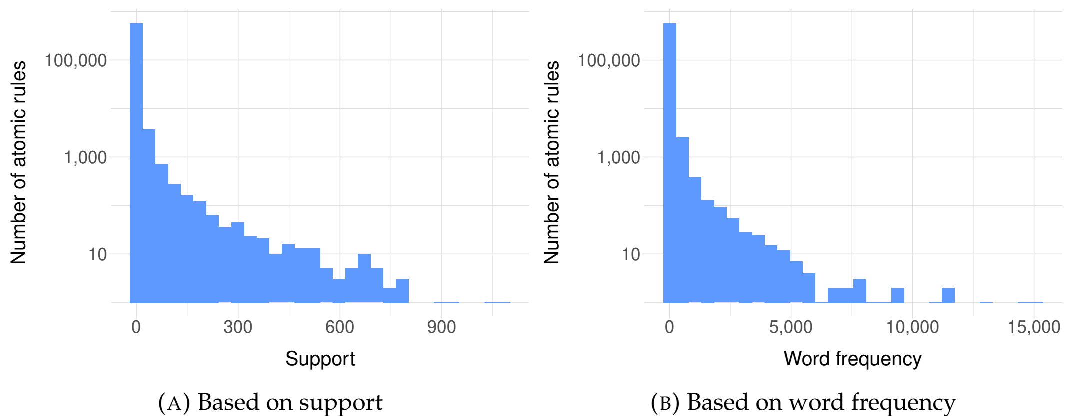


(A) Average number of responses (B) Average expected response index

FIGURE 6.2: The average number of responses and the average index of the expected response based on the number of retained atomic rules using p_{supp} and p_{freq} optimization

index of the expected response slightly increases to about 1.7 and 1.8 from 1.5, which are still acceptable.

Figure 6.3 displays the histogram of the atomic rules based on their support values and word frequencies. As we can see, choosing a relatively low support or word frequency threshold will eliminate the majority of the atomic rules.



(A) Based on support (B) Based on word frequency

FIGURE 6.3: The histogram of the number of atomic rules based on their support values and word frequencies

In Table 6.1 I collected some of the $(p_{gen},\ p_{max})$ parameter combinations and the resulting metric values. From the table it is obvious that $(p_{gen} = 1,\ p_{max} = 1)$ is not an optimal configuration, since the model overgeneralizes: both the number of responses and the index of the expected response become the highest, while the average accuracy also drops to about 85.78%. Those rows that produce an acceptable average accuracy and a relatively low number of responses, do not eliminate many atomic rules, so p_{supp} and p_{freq} seem to have better results.

The $(p_{gen} = 2, p_{max} \in \{4, 5\})$ configurations seem to be almost equal if not better than $p_{supp} = 10$, but they resulted in slower inflection and analysis times. Since the p_{supp} based optimization also gets rid of the too specific rules (as the support values of the covered atomic rules are additive), it seems like using the support value instead of the number of retained atomic rules per word pair gives a better rule base to work with.

DOI: 10.14750/ME.2022.020

TABLE 6.1: The average number of retained atomic rules, correctness ratio, number of responses and expected response index using different $(p_{gen},\ p_{max})$ combinations

p_{gen}	p_{max}	**Rules**	**Accuracy**	**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
3	5	302,497	82.57%	1.60	1.25

6.4 Evaluation

This section consists of two parts. In Subsection 6.4.1 I evaluate the winning optimization parameter $(p_{supp} = 10)$ and compare its results with the results of the unoptimized Morpher model in Section 5.6. I use the same metrics, training and evaluation data sizes and evaluation methodology.

I also wanted to evaluate $(p_{gen} = 1,\ p_{max} = 1)$ but it proved to be unacceptably slow using bigger training data sets, due to the decreased information of the system.

In Subsection 6.4.2 I use a training data set of 500,000; 1,000,000; ...; 3,000,000 training items to see how well the optimized Morpher model scales. I measure the same metrics as before: the average training time, size, inflection time, analysis time, accuracy, number of responses and index of the expected response.

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

6.4.1 Comparison with the Baseline Morpher Model

Average Training Time

In Figure 6.4 we can see the average training time of the baseline (unoptimized) and the optimizied Morpher model using $p_{supp} = 10$. The chart shows that using 100,000 training items, the optimization parameter reduced the average training time from about 3.9 seconds to about 2.9 seconds. The cause of this might be that less atomic rules need to be managed and stored.

Average Size

The average size of the unoptimized and the optimized Morpher model is displayed in Figure 6.5. Since I only reduced the rule base size, the average number of atomic rules is measured, and due to the big differences, I use logarithmic scale on the y axis.

DOI: 10.14750/ME.2022.020

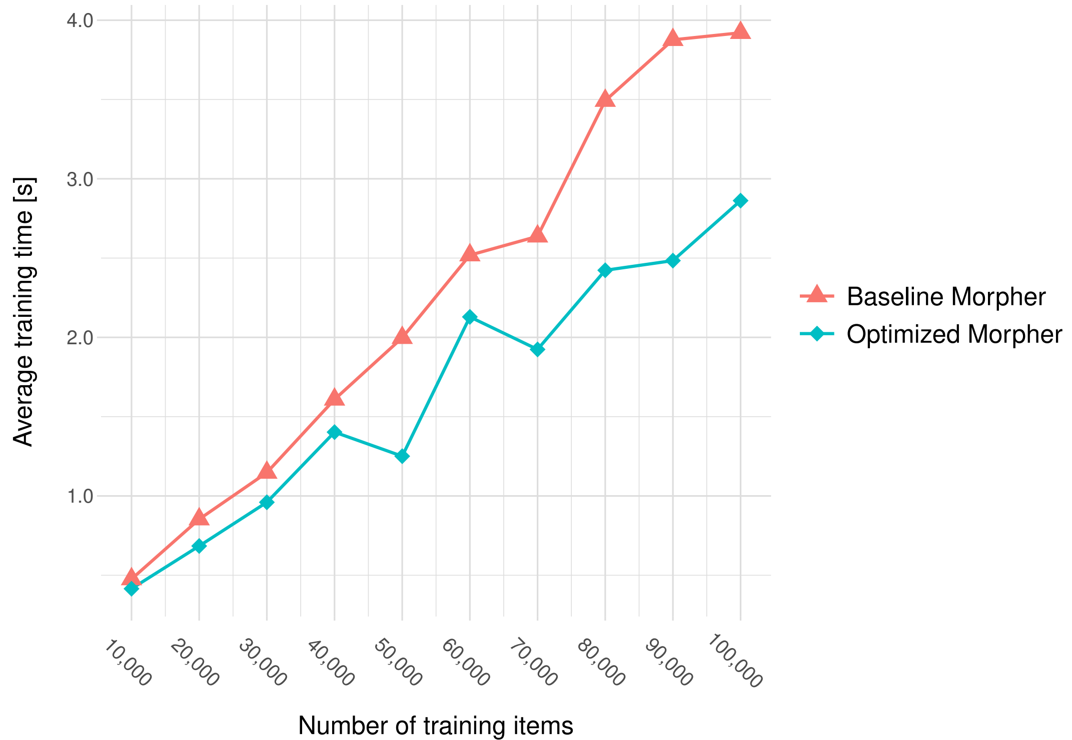


FIGURE 6.4: The average training time of the baseline and the optimized Morpher model

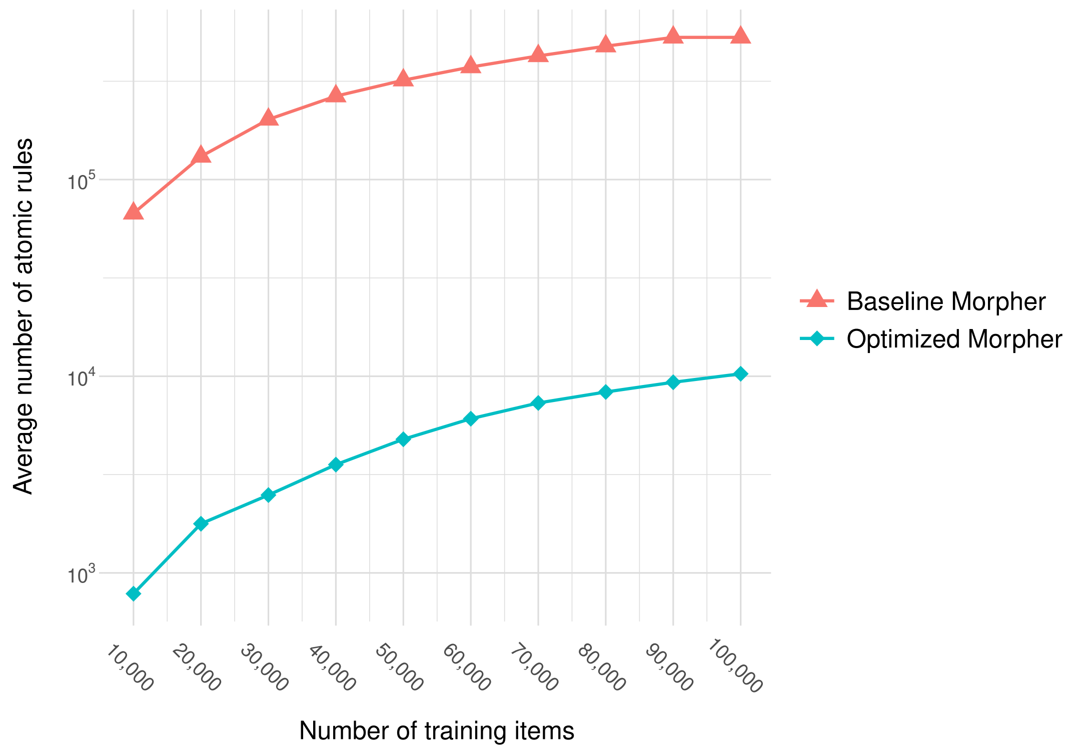


FIGURE 6.5: The average size of the baseline and the optimized Morpher model

DOI: 10.14750/ME.2022.020

Using 100,000 training items, the optimized Morpher model retained only 10,291 atomic rules, compared to the 578,497 rules of the baseline model. This means that the optimized model contains only about 1.78% of the complete rule base.

The file size of the model is also reduced from 8.5 MB to 5.2 MB. The reduction in file size is less, since the exported file also contains the lemmas and conditional probabilities besides the atomic rules.

Average Inflection and Analysis Time

Figure 6.6 is about the average inflection time and the average morphological analysis time of the baseline and the optimized Morpher model, using logarithmic scale on the y axis.

We can see that for both operations, applying the $p_{supp} = 10$ optimization parameter has a positive effect. The average inflection time became 0.7 milliseconds from 2.4 milliseconds, while the average analysis time became 21.75 milliseconds from 2.4 seconds.

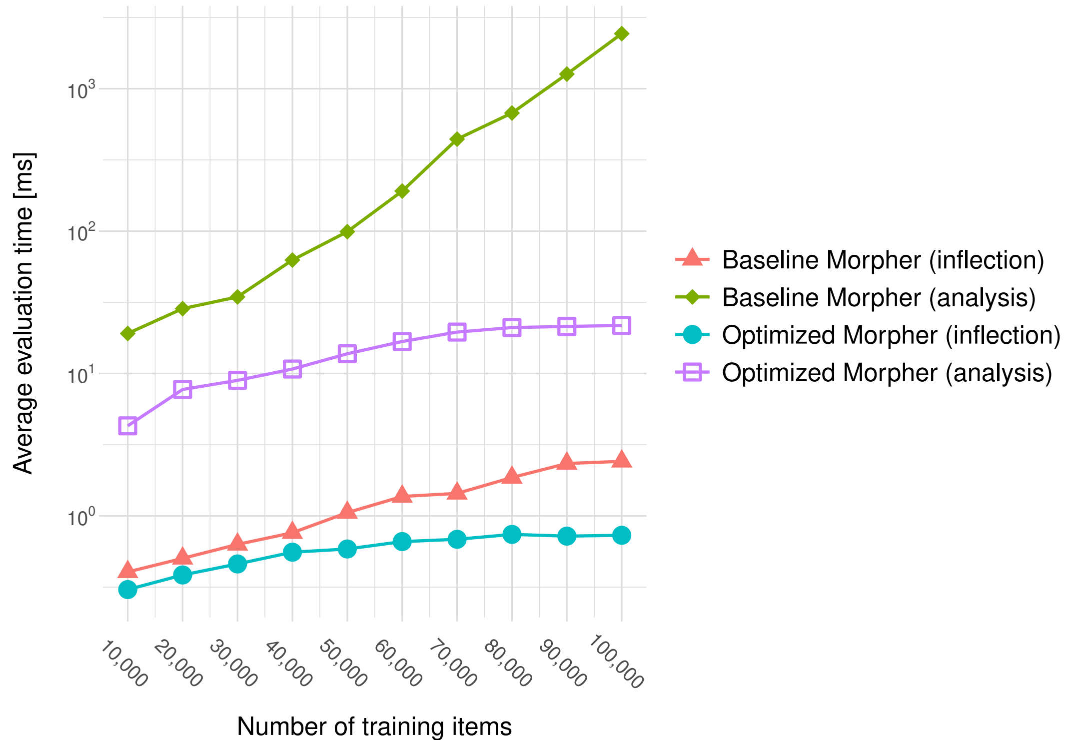


FIGURE 6.6: The average inflection time and the average analysis time of the baseline and the optimized Morpher model

This means an almost 70% reduction in average inflection time and an about 99.1% reduction in case of the average analysis time.

Another interesting observation is that the lines of the optimized metrics are less steep, meaning that the differences would increase if we further increased the training data set size.

DOI: 10.14750/ME.2022.020

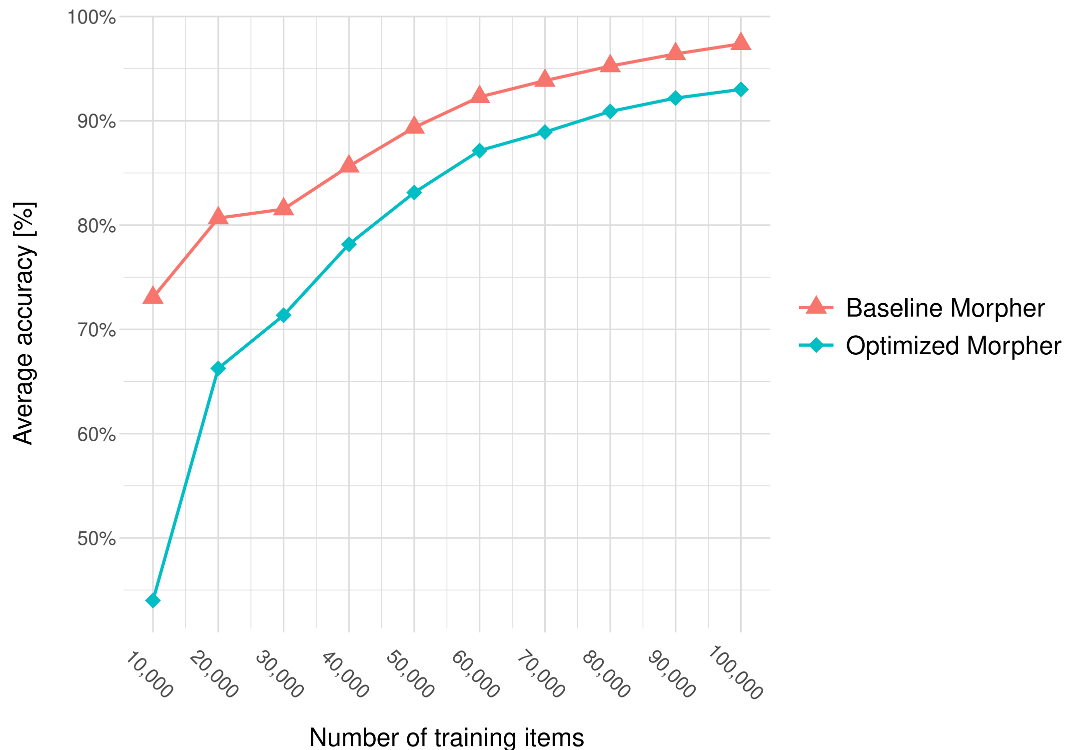


FIGURE 6.7: The average accuracy of the baseline and the optimized Morpher model

Average Accuracy

Figure 6.7 displays the average accuracy differences between the baseline and the optimized Morpher model. Unfortunately, performing a p_{supp} based atomic rule base reduction will reduce the overall average accuracy, too, but the resulting average accuracy using 100,000 training items is still about 93.01% which is acceptable. And although the unoptimized model performs better, we can see that the curves still increase, meaning that increasing the size of the training data set will result in higher values.

Using 100,000 training items, the average number of responses is reduced from 37 to 9.8 in case of inflection (which means an almost 75% reduction) and from 5.4 to 3.4 in case of morphological analysis (about 37% reduction).

As for the average index of the expected response, it increased a bit in case of inflection, from 1.5 to about 1.9, and stayed around 2.4 in case of morphological analysis.

6.4.2 Using Big Training Data Volumes

Average Training Time

Figure 6.8 shows the average training time of the optimized Morpher model using $p_{supp} = 10$ and big training data volumes. As we can see, the average training time increases roughly linearly, reaching about 74.61 seconds using 3 million training items.

DOI: 10.14750/ME.2022.020

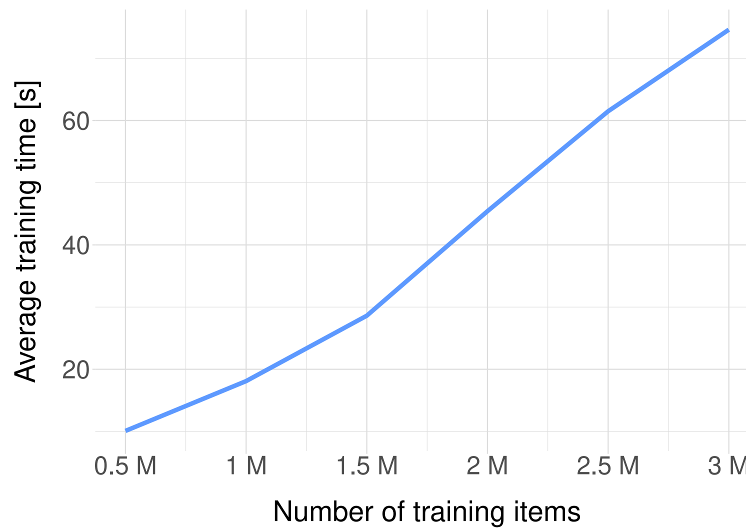


FIGURE 6.8: The average training time of the optimized Morpher model using big training data volumes

Average Size

In Figure 6.9 we can see the number of generated atomic rules of the optimized Morpher model. Similarly to the average training time, the number of atomic rules also increases about linearly. Using 3 million training items, there are 255,867 atomic rules generated by Morpher. The file size of the exported knowledge base is 5.5 MB.

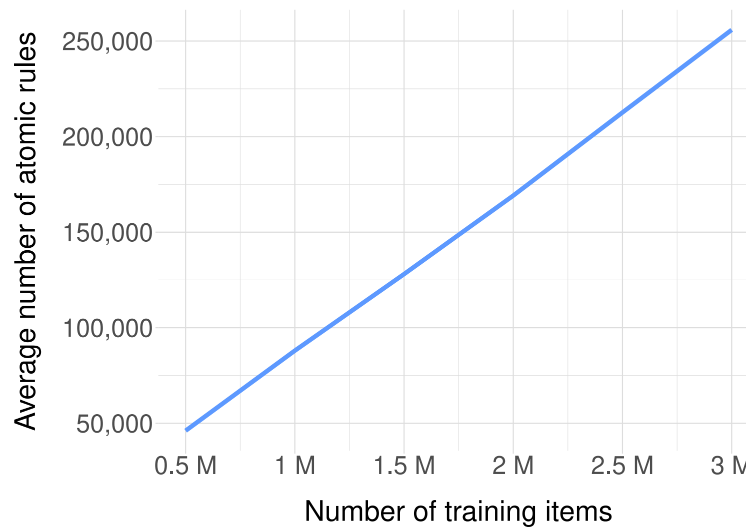
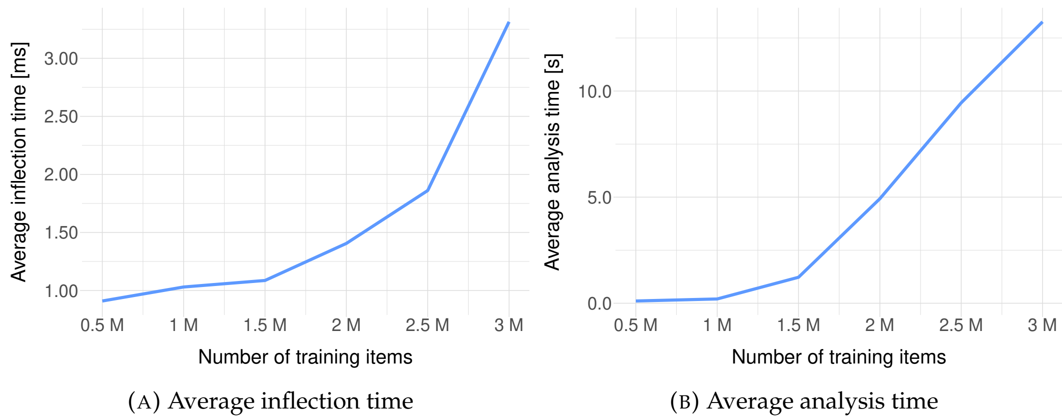


FIGURE 6.9: The average number of atomic rules of the optimized Morpher model using big training data volumes

Average Inflection and Analysis Time

Figure 6.10 displays the average inflection time (left side) and the average analysis time (right side) of the optimized Morpher model. The two operations still differ by orders of magnitude, but the scale of the increase, as well as the values are lower

DOI: 10.14750/ME.2022.020



(A) Average inflection time (B) Average analysis time

FIGURE 6.10: The average inflection and analysis time of the optimized Morpher model using big training data volumes

than before. (Without any optimizations, Morpher would not be able to handle these operations in acceptable, finite time in case of big training data sets.)

After training Morpher with 3 million training items, inflection took about 3.3 milliseconds in average, while morphological analysis took about 13.3 seconds. This is a higher value, but we can still say that the model could finish the analysis in finite time, which is a big step forward, compared to the results of the unoptimized model.

Average Accuracy

In Figure 6.11 we can see the average accuracy of the optimized Morpher model using big training data volumes. Even though using 100,000 training items the average accuracy seemed to drop off a couple of percents, further increasing the size of the training data set we can see an increase between 96.22% and 98.11%. However, the average accuracy seems to be plateauing around this value, beyond about 1 million training items.

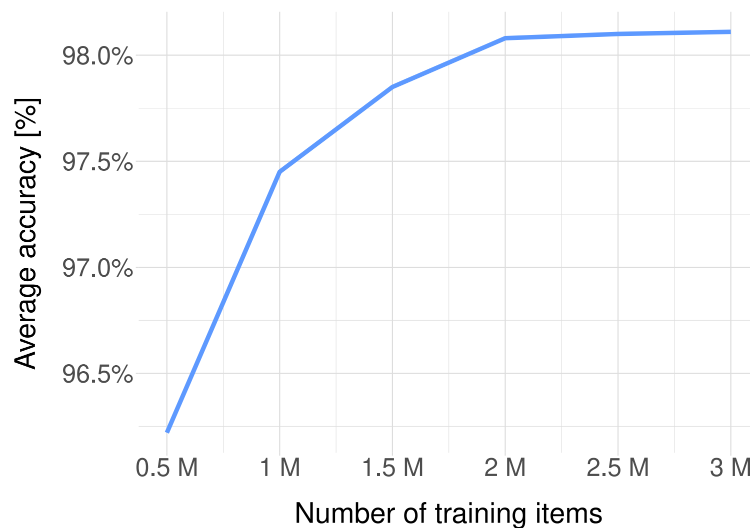
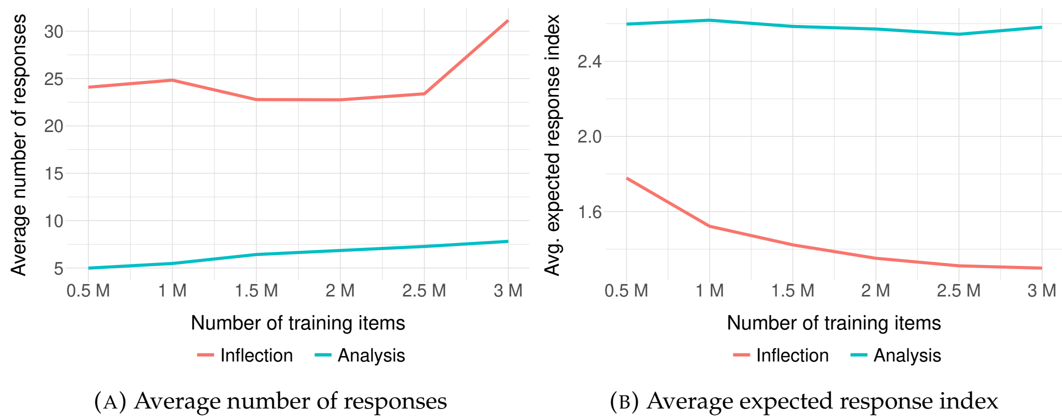


FIGURE 6.11: The average accuracy of the optimized Morpher model using big training data volumes

DOI: 10.14750/ME.2022.020



(A) Average number of responses (B) Average expected response index

FIGURE 6.12: The average number of responses and the average index of the expected response of the optimized Morpher model using big training data volumes

Figure 6.12 shows the average number of responses (left side) and the average index of the expected response (right side). In the left chart, we can see that the final value of the average number of responses is about 31.1 for inflection and 7.8 for morphological analysis. This means that even using such huge training data sets, this metric will be lower in case of inflection than for the unoptimized Morpher model using 100,000 training items (37).

In the right-side image we can see that the average index of the expected response became about 1.2 in case of inflection and 2.5 in case of morphological analysis. This means that while using 3 million training items, the correct response almost always had one of the highest aggregated weights in the list provided by Morpher.

6.5 Conclusion

In this chapter I performed the space complexity analysis of the main components of the Morpher model, including ASTRA, as well as the time complexity analysis of the main steps of the training phase, inflection generation and morphological analysis operations of the model.

After analyzing the space and time complexity of Morpher, I proposed three optimization techniques to reduce the atomic rule base:

- Redundant atomic rule elimination (p_{max}): eliminates the atomic rules that make the rule base more redundant
- Limiting the generalization factor (p_{gen}): eliminates the atomic rules that potentially contribute to the overgeneralization effect
- Indirect noise reduction (p_{supp} and p_{freq}): eliminates the weaker atomic rules whose support value or word frequency is lower than a given threshold

Analyzing these optimization parameters using up to 100,000 training items, it turned out that the $p_{supp} = 10$ model configuration achieved the best results: its average accuracy remained high, while the model still provided its responses confidently, despite its rule base decreased to about the 1.78% of the original rule base. Table 6.2 summarizes the differences of the metrics using p_{supp} optimization and 100,000 training items.

Although the optimization made the model usable with up to 3 million training items, from the resulting evaluation data we can see that above 1 million there is no

DOI: 10.14750/ME.2022.020

real benefit of further expanding the size of the training data set. Table 6.3 contains the metric values of the optimized Morpher model using 1 million training items.

TABLE 6.2: Summary of the measured metrics of the baseline and the optimized Morpher model using 100,000 training items

Metric	**Baseline**	$p_{supp} = 10$	**Difference**	**Improvement?**
Training time	3.9 s	2.9 s	−25.64%	✓
Rules	578,497	10,291	−98.22%	✓
Knowledge base	8.5 MB	5.2 MB	−38.82%	✓
Inflection time	2.4 ms	0.7 ms	−70.83%	✓
Analysis time	2.4 s	21.75 ms	−99.09%	✓
Accuracy	97.38%	93.01%	−4.49%	✗
Inflection responses	37	9.8	−73.51%	✓
Analysis responses	5.4	3.4	−37.04%	✓
Inflection index	1.5	1.9	+26.67%	✗
Analysis index	2.4	2.4	+0%	

TABLE 6.3: Summary of the measured metrics of the optimized Morpher model using 1 million training items

Metric	**Value**
Training time	18.1 s
Rules	88,005
Knowledge base	3.9 MB
Inflection time	1.03 ms
Analysis time	0.2 s
Accuracy	97.45%
Inflection responses	24.8
Analysis responses	5.47
Inflection index	1.52
Analysis index	2.62

DOI: 10.14750/ME.2022.020

Thesis 4 ***[3] [4] [5]***

I have performed the space and time complexity analysis of the Morpher and ASTRA models. After analyzing the required space of the model components and the required time of the training phase, inflection generation and morphological analysis operations, I have proposed three optimization techniques that aim to reduce the rule base of the model and thus decrease the average inflection and analysis times. Evaluation shows that using the same amount of training data, the number of retained rules dropped from 578,497 to 10,291, the average inflection time decreased from 2.4 ms to 0.7 ms, and the average analysis time was reduced from 2.4 s to 21.75 ms, while the average accuracy remained 93.01%. Finally I evaluated the optimized Morpher model using up to 3 million training items, and both inflection generation and morphological analysis could be performed in acceptable, finite time, even though the unoptimized original model could not handle such big training data volumes at all.

DOI: 10.14750/ME.2022.020

Chapter 7

The Reference Implementation of the Morpher Ecosystem

In this chapter I describe the training and evaluation data generation process (Section 7.1) and the implementation of the Morpher framework, the Morpher API and the Morpher client (Section 7.2).

7.1 The Training and Evaluation Data Generation Process

To generate large volumes of training and evaluation data for the evaluation of the Morpher framework, I first collected PDF documents from the website of the Hungarian Electric Library,[1] and extracted every unique word out of these documents. To speed up the word extraction process, I used Java parallel streams to process each page of the website in parallel. For the PDF document processing, I used the iText Java library.[2] The number of collected documents was 16,250; while the number of the extracted word candidates was 13,345,903.

For the morphological analysis of these word candidates, I used Hunmorph-Ocamorph, that resulted in 4,423,882 different morphological structures for 2,515,570 unique word forms. The provided morphological structures include the lemma, part of speech and morphosyntactic tags.

Since I used Hunmorph-Ocamorph to generate the training and evaluation data sets, the knowledge base of the proposed model is a subset of that of the source model. This is not a problem, because the goal of this research project was to experiment with more compact data structures and alternative learning methods that can extract the necessary information in a more optimized way, reaching a higher generalization factor. To achieve these goals, using Hunmorph-Ocamorph I could generate large amounts of data in an automated way.

The data set of records containing an inflected word form, a lemma, a part of speech and a list of affix type tags was sufficient to evaluate Morpher itself, since it can deduce training word pairs for the transformation engine instances internally. However, evaluating the single-affix transformation engine models required appropriate word pair sets demonstrating the transformation rules of each Hungarian affix type. In the experiments, I chose the Hungarian accusative case to evaluate these models.

The algorithm I used to generate the word pairs is basically the same that I ported to the training model of Morpher, as described by Section 5.3. The algorithm has the following main steps:

1. Group the records based on their lemmas.

[1] `https://mek.oszk.hu`
[2] `https://itextpdf.com`

DOI: 10.14750/ME.2022.020

2. Process each group, starting from the word containing the most affix types.
3. If two records are found that have the same lemma and the same affix types, except that the first record has an extra affix type at the end of the list, generate a word pair from the two inflected word forms.
4. If a record is found that has only one affix, generate a word pair containing its lemma and its inflected form.

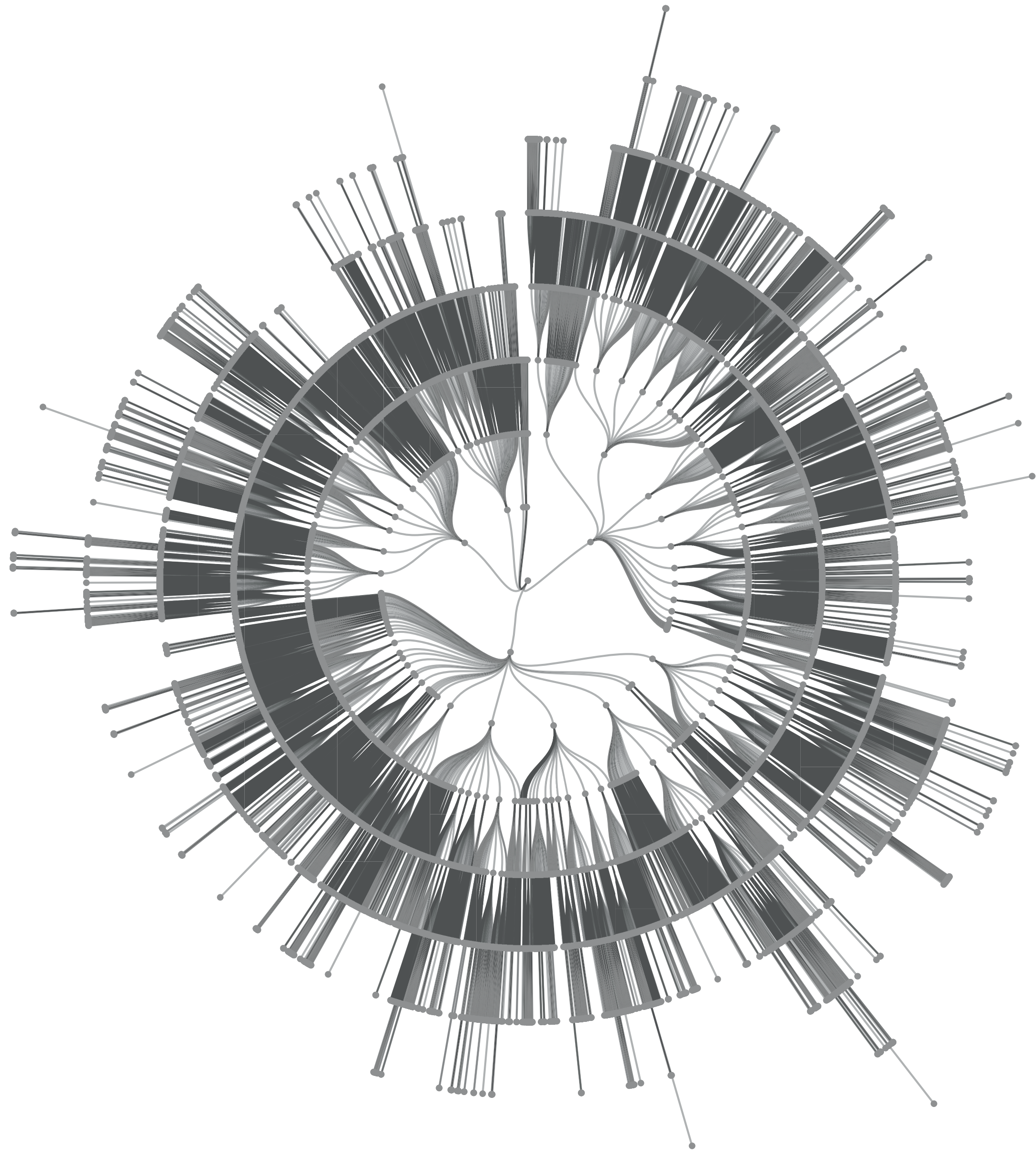


FIGURE 7.1: The visualization of the valid affix type chains in the Hungarian language based on the generated data set

Example 7.1 (Generating word pairs)**.** *Let us take the following responses of Hunmorph:*[3]

1. *almák → alma/NOUN⟨PLUR⟩*
2. *labdák → labda/NOUN⟨PLUR⟩*
3. *almákat → alma/NOUN⟨PLUR⟩⟨CAS⟨ACC⟩⟩*
4. *labdát → labda/NOUN⟨CAS⟨ACC⟩⟩*

[3]The responses relate to the inflected forms of *alma* (*apple*) and *labda* (*ball*).

DOI: 10.14750/ME.2022.020

5. *almát → alma/NOUN⟨CAS⟨ACC⟩⟩*

After grouping, items 1, 3 and 5 will be part of the same group, while items 2 and 4 will be in the other group.

Since the item 1, 2, 4 and 5 only contain one affix type, we can directly generate word pairs from them: (alma, almák) and (labda, labdák) for plural, as well as (labda, labdát) and (alma, almát) for accusative case.

From item 3, we can only generate a word pair if we find a record whose lemma is also alma, and that only contains the affix type ⟨PLUR⟩. Fortunately, item 1 is exactly such a record, so we can use its inflected form as the base form of the new word pair: (almák, almákat) for accusative case.

The total number of generated word pairs was 3,625,036. The whole generated data set is stored on Github.[4]

The complexity of the Hungarian affix type system can be seen in Figure 7.1, where I visualize all the valid affix type chains based on the generated data. Each dot in the figure is an affix type, and the edges denote their adjacency. There are 35,954 different affix type chains, the longest ones containing 9 affix types. The median of the affix type chain lengths is 5.

7.2 The Layers of the Morpher Ecosystem

The implemented Morpher ecosystem consists of several projects:

- The Morpher framework (Subsection 7.2.1) is a Java based framework that implements the Morpher model, including the transformation engine models.
- The Morpher API (Subsection 7.2.2) is a server-side Spring Boot based REST API that publishes the inflection generation and morphological analysis operations of a pre-trained Morpher engine instance.
- The Morpher client (Subsection 7.2.3) is a React and React Native based web and mobile client application that provides user interface for inflection generation and morphological analysis, consuming the Morpher REST API endpoints.

The Morpher ecosystem has a client-server architecture, as displayed in Figure 7.2: the client application can run in a web browser or on a mobile phone, and connects to the Morpher API that uses the Morpher framework and loads a pre-trained model file during initialization.

7.2.1 Morpher Framework

The Morpher framework is a modular morphology framework that can solve the inflection generation and morphological analysis problems. The framework can be found on Github,[5] and the compiled binaries are published on jcenter[6] and Maven Central.[7]

The programming platform of the Morpher framework is Java version 17 as of writing, that provides a module system that Morpher utilizes, as well as the possibility to process large collections in parallel using parallel streams. The serialization and deserialization of the objects are performed using protocol buffers.[8]

[4] https://github.com/szgabsz91/morpher-data
[5] https://github.com/szgabsz91/morpher
[6] https://bintray.com/search?query=morpher
[7] https://search.maven.org/search?q=morpher
[8] https://developers.google.com/protocol-buffers

DOI: 10.14750/ME.2022.020

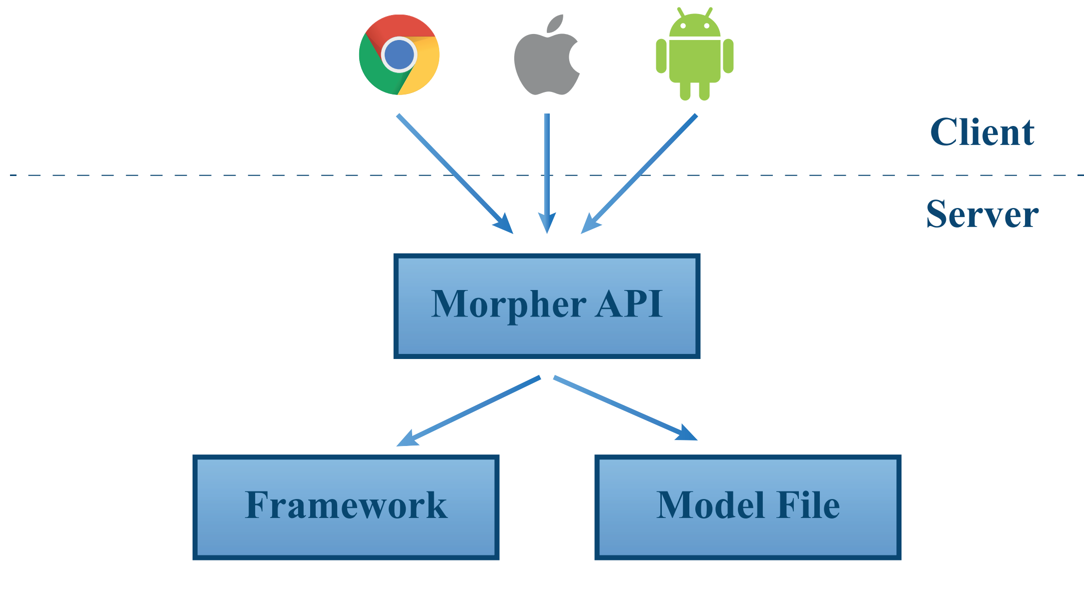


FIGURE 7.2: The architecture of the Morpher ecosystem

Morpher Core

The Morpher Core submodule forms the basis of the Morpher model. It contains

- converter interfaces for converting objects to and from protocol buffers,
- serialization and deserialization related interfaces,
- I/O related classes to load from and save to disk,
- simple POJO classes to model affix types, corpora, words, word pairs, etc., and
- utility classes to retrieve service instances from the Java module system based on custom qualifiers.

Morpher Transformation Engines

There are five pre-implemented transformation engines in the Morpher framework repository:

- the dictionary based transformation engine that uses standard Java `HashMap`s,
- the FST based transformation engine that uses the Lucene[9] FST implementation,
- the TASR based transformation engine that uses a custom TASR implementation,
- the lattice based transformation engine that implements the model introduced in Section 4.1, and
- the ASTRA based transformation engine that implements the model introduced in Section 4.2.

All of these transformation engine implementations implement the same API. In the `morpher-transformation-engine-api` submodule there is an interface called `IBidirectionalTransformationEngine`, that lists all the required operations of the transformation engines. The most important methods can be seen in Listing 1.

The `IAbstractTransformationEngineFactory` interface is a link between the previous interface and the Java module system. Using the `ServiceLoader`

[9] `https://lucene.apache.org`

DOI: 10.14750/ME.2022.020

Listing 1 The `IBidirectionalTransformationEngine` interface

```
public interface IBidirectionalTransformationEngine {

    AffixType getAffixType();

    int size();

    void learn(TrainingSet trainingSet);

    Optional<TransformationEngineResponse> transform(Word w);

    Optional<TransformationEngineResponse> transformBack(Word w);

}
```

class provided by Java, it is very easy to retrieve an abstract transformation engine factory instance, that can then supply the bidirectional transformation engine instance based on the provided configuration parameter values. This way the transformation engine implementation class can be hidden from the outer world.

Morpher Language Handlers

The language dependent components of the Morpher engine are organized into their own submodules. The `ILanguageHandler` interface provides methods to

- learn affix type chains,
- calculate the conditional probabilities of the affix type chains,
- learn lemmas, and
- determine the next affix type candidates during inflection generation and morphological analysis.

There is currently only one language handler implementation that is based on Hunmorph-Ocamorph. This service instance is provided by the Java module system, similarly to the transformation engine implementations.

Morpher Engines

There are currently two Morpher engine implementations: one is totally based on Hunmorph-Ocamorph and thus can only analyze the input words, while the other one implements the Morpher model of Chapter 5.

The main methods of the `IMorpherEngine` interface can be seen in Listing 2: it can learn the morphological specialties of a language from several data structures (corpora, preanalyzed training items containing every information or maps containing lemmas and their parts of speech), then it can inflect and analyze words, and return the supported affix types of the language.

Instantiating a Morpher engine instance can be easily done using the Java module system. We only need to provide the unique qualifier of the transformation engine implementation to use, its configuration parameters, and the unique qualifier of the language handler implementation to use. The related Java code (that extensively uses the builder design pattern) can be seen in Listing 3.

Listing 4 shows the Java code that we can use to have the Morpher engine instance learn, inflect and analyze.

DOI: 10.14750/ME.2022.020

Listing 2 The `IMorpherEngine` interface

```
public interface IMorpherEngine {

    void learn(Corpus corpus);

    void learn(PreanalyzedTrainingItems trainingItems);

    void learn(LemmaMap lemmaMap);

    List<MorpherEngineResponse> inflect(InflectionInput input);

    List<MorpherEngineResponse> analyze(AnalysisInput input);

    List<AffixType> getSupportedAffixTypes();

}
```

Listing 3 Creating the Morpher engine instance

```
var serviceProvider = new ServiceProvider(
    clazz -> ServiceLoader.load(clazz).stream()
);
var config = new ASTRATransformationEngineConfiguration.Builder()
    .searcherType(SearcherType.PARALLEL)
    .build();
var engine = new MorpherEngineBuilder<>()
    .serviceProvider(serviceProvider)
    .transformationEngineQualifier(
        IASTRATransformationEngine.QUALIFIER
    )
    .transformationEngineConfiguration(config)
    .languageHandlerQualifier(IHunmorphLanguageHandler.QUALIFIER)
    .build();
```

Listing 4 Using the Morpher engine instance

```
engine.learn(Corpus.of(Word.of("almát")));

var inflectionInputWord = Word.of("alma");
var inflectionInputAffixTypes = Set.of(AffixType.of("<CAS<ACC>>"));
var inflectionInput = new InflectionInput(
    inflectionInputWord,
    inflectionInputAffixTypes
);
var inflectionResponses = engine.inflect(inflectionInput);

var analysisInputWord = Word.of("almát");
var analysisInput = AnalysisInput.of(analysisInputWord);
var analysisResponses = engine.analyze(analysisInput);
```

DOI: 10.14750/ME.2022.020

7.2.2 Morpher API

Since the Morpher framework is developed in Java, it can be consumed by Java applications easily. However, integration can become quite complex using other ecosystems like .NET or Node.js.

This is why I created the Morpher API,[10] that is a Spring Boot based REST API application, publishing the inflection generation and morphological analysis operations of a pre-trained Morpher engine instance over the HTTP protocol.

This web application has three main REST endpoints:

- `.../affix-types`: returns the list of supported affix types of the underlying Morpher engine instance
- `.../inflect?input&affix-types`: inflects the given lemma using the provided comma-separated set of affix types
- `.../analyze?input`: analyzes the provided word

The web application is published as a Docker image.[11]

[10] `https://github.com/szgabsz91/morpher-api`
[11] `https://hub.docker.com/r/szgabsz91/morpher-api`

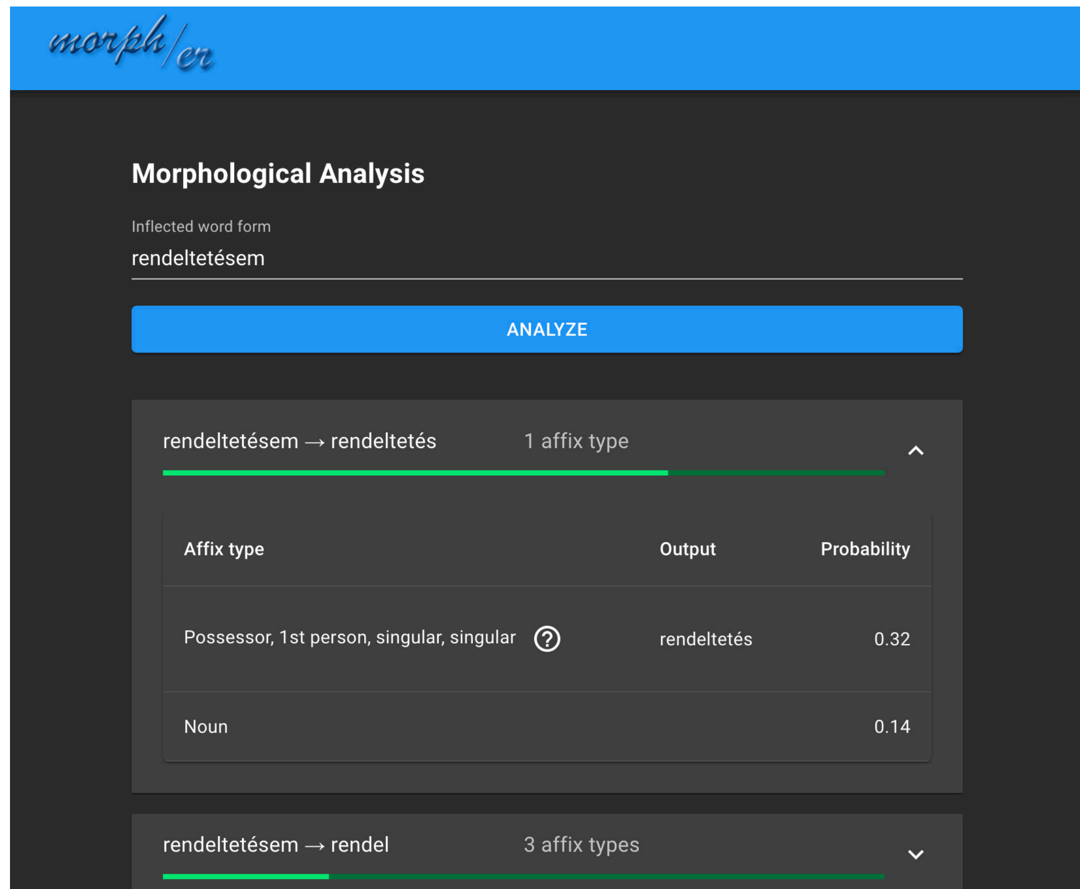


FIGURE 7.3: The Morpher web client

DOI: 10.14750/ME.2022.020

7.2.3 Morpher Client

The Morpher client application[12] is a React and React Native based client application, containing three submodules managed by Lerna:[13]

- `shared`: shared module that contains i18n related code, localized labels, services to dispatch HTTP requests and common assets
- `web`: the web application developed using React
- `mobile`: the mobile application developed using React Native that can run on both Android and iOS devices

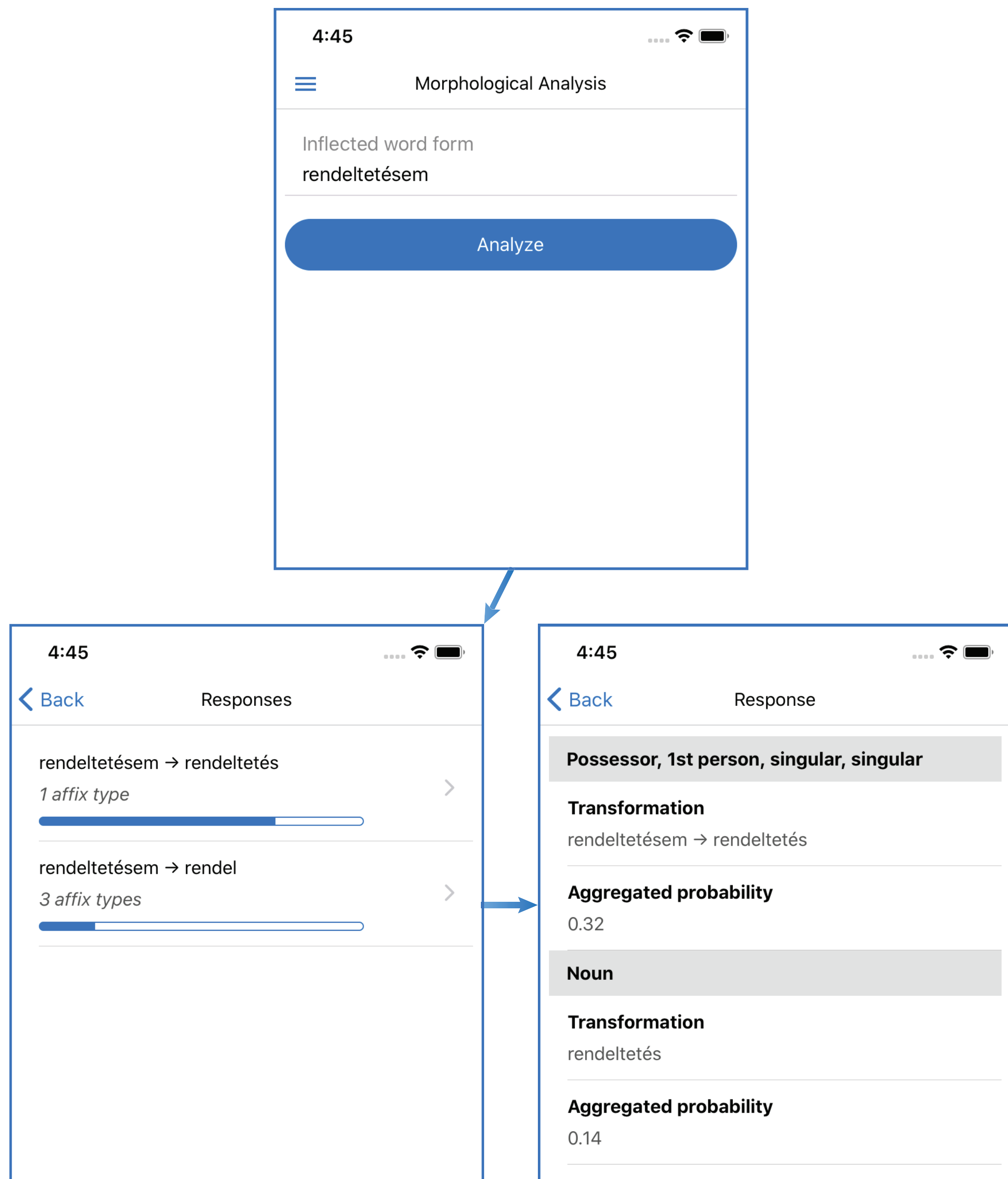


FIGURE 7.4: The Morpher mobile client

[12] https://github.com/szgabsz91/morpher-client
[13] https://lerna.js.org

DOI: 10.14750/ME.2022.020

The client application consumes the REST endpoints of the Morpher API, and provides user interface for the inflection generation and morphological analysis operations.

In the web client, the responses are displayed in an accordion. The header contains the input and output, as well as the aggregated weight in the form of a progressbar. In case of morphological analysis, the number of affix types found in the input word is also displayed. After the user opens a row, its steps are displayed in a table. In case of inflection generation, the part of speech is contained by the first row, while in case of morphological analysis, it is the last row in the table. In Figure 7.3 we can see a sample screenshot of the web based morphological analysis page.

The mobile client is slightly different in that the responses and the table of steps are displayed in separate screens instead of in an accordion to support smaller screen sizes. A sample morphological analysis workflow can be seen in Figure 7.4.

The web client is published as a Docker image.[14] The production build is served by NGINX.[15] The published Docker image can be used alongside the Morpher API Docker image using Docker Compose.[16]

7.3 Conclusion

In this chapter I described the automated training and evaluation data generation process (Section 7.1), as well as the layers of the Morpher ecosystem (Section 7.2).

The data source of the automated training and evaluation data generation process was the website of the Hungarian Electric Library, from where I first downloaded PDF documents, then parsed them and extracted the unique word candidates out of them. These word candidates have been processed using Hunmorph-Ocamorph to get a set of morphologically analyzed words. From these records, I finally generated word pairs for each Hungarian affix type for the evaluation of the single-affix transformation engine models. The total number of morphological structures was almost 4.5 million, from which I could generate more than 3.5 million word pairs in total.

The implementation of the Morpher framework, the Morpher REST API and the Morpher client application can be found on Github. The built binaries of the Morpher framework are published on jcenter and Maven Central, while the Morpher API and the Morpher client application are published as Docker images on Docker Hub. The Morpher framework and the Morpher API are developed in Java using modularization features extensively, while the Morpher client application is written using React and React Native.

The proposed Morpher model can be used by higher-level grammatical models such as syntactic analyzers or free text processing frameworks. Thanks to the Morpher API project, inflection generation and morphological analysis can be invoked from virtually any programming environments.

Thesis 5 ***[1] [3] [4] [5] [8]***

I have developed the reference implementation of the Morpher ecosystem, that consists of several layers such as the Morpher multi-affix morphology model and the single-affix transformation engine models including the lattice based model and ASTRA. I have also developed a Spring Boot based Morpher REST API, so that the

[14] `https://hub.docker.com/r/szgabsz91/morpher-client`
[15] `https://www.nginx.com`
[16] `https://docs.docker.com/compose`

DOI: 10.14750/ME.2022.020

Morpher framework can be consumed more easily from different software environments. To provide web and mobile user interfaces for inflection generation and morphological analysis, I have developed the Morpher client application using React and React Native. For training and evaluation purposes, I have also implemented an automated method to generate large training and evaluation data sets for the Hungarian language, resulting in more than 4.4 million morphological structures, covering more than 2.5 million unique word forms. The source code of these projects can be found on Github, while the binaries are published to jcenter and Maven Central, and the Docker images are published to Docker Hub.

DOI: 10.14750/ME.2022.020

Chapter 8

Conclusion

8.1 Contribution

In this dissertation I presented my contribution in the automated learning of inflection generation and morphological analysis. The proposed models use classical pattern matching based methods instead of artificial intelligence.

The problem is approached on two levels: first, I proposed two single-affix transformation engine models, then the multi-affix morphology model called Morpher. The responsibility of the single-affix transformation engine models is to learn the transformation rules of a single affix type from a provided set of training word pair set. On the other hand, the Morpher model coordinates the work of several transformation engine model instances (one for each affix type of the target language) to solve the inflection generation and morphological analysis problems, handling all the affix types of the target language. The proposed models were evaluated using Hungarian training and evaluation data sets, comparing them with state of the art morphology models from the literature, many of which use some kind of neural network architecture.

The novel scientific results can be summarized by the following five theses.

Thesis 1 **[1]**

I have designed and implemented a new method to compare, analyze, evaluate and rank morphological analyzers. This method is based on novel formulae to calculate similarity and distance values among the different analyzers, including the recognition similarity, token similarity, mapping similarity; as well as the recognition distance, token distance, mapping distance and cumulative distance. I applied this analysis method on four popular morphological analyzers of the Hungarian language, namely Hunmorph-Ocamorph, Hunmorph-Foma, Humor and Hunspell. For the evaluation, I created a token mapping among these analyzers as well. Based on the performed evaluation, Hunmorph-Ocamorph proved to be the most usable model among the four analyzers.

Thesis 2 **[2] [3] [9] [10] [11] [12] [13] [14] [15] [16]**

I have proposed two novel single-affix transformation engine models that can learn inflection rules from a provided set of training word pairs. The first one is a lattice based model that has a more complex, position dependent rule structure, and stores its rules in a lattice. The second model called ASTRA describes inflection as a set of simple string transformations, omitting the position indices from its rule model. The atomic rules are stored in either a set or a prefix tree based data structure. Both models apply pattern matching during the rule search process. I performed

DOI: 10.14750/ME.2022.020

the evaluation of the proposed models, showing that while the lattice based model can achieve minimal storage size, the ASTRA model has an outstanding accuracy for previously unseen words (about 94%), beating the examined baseline models including TASR, FST and a dictionary implementation.

Thesis 3 *[3] [4]*

I have proposed a novel multi-affix morphology model called Morpher that can solve the inflection generation and morphological analysis problems, handling all the affix types of the target language. The main feature of the proposed Morpher model is that it builds a separate transformation engine instance for each affix type, and it takes the conditional probabilities of the affix type chains into account during inflection generation and morphological analysis. During the evaluation of Morpher I used the ASTRA model to train the transformation engines. The experiments confirmed the outstanding generalization capabilities and accuracy of Morpher, comparing it with state of the art models including 6 SIGMORPHON models, 3 unsupervised segmentation models and 2 analyzer models.

Thesis 4 *[3] [4] [5]*

I have performed the space and time complexity analysis of the Morpher and ASTRA models. After analyzing the required space of the model components and the required time of the training phase, inflection generation and morphological analysis operations, I have proposed three optimization techniques that aim to reduce the rule base of the model and thus decrease the average inflection and analysis times. Evaluation shows that using the same amount of training data, the number of retained rules dropped from 578,497 to 10,291, the average inflection time decreased from 2.4 ms to 0.7 ms, and the average analysis time was reduced from 2.4 s to 21.75 ms, while the average accuracy remained 93.01%. Finally I evaluated the optimized Morpher model using up to 3 million training items, and both inflection generation and morphological analysis could be performed in acceptable, finite time, even though the unoptimized original model could not handle such big training data volumes at all.

Thesis 5 *[1] [3] [4] [5] [8]*

I have developed the reference implementation of the Morpher ecosystem, that consists of several layers such as the Morpher multi-affix morphology model and the single-affix transformation engine models including the lattice based model and ASTRA. I have also developed a Spring Boot based Morpher REST API, so that the Morpher framework can be consumed more easily from different software environments. To provide web and mobile user interfaces for inflection generation and morphological analysis, I have developed the Morpher client application using React and React Native. For training and evaluation purposes, I have also implemented an automated method to generate large training and evaluation data sets for the Hungarian language, resulting in more than 4.4 million morphological structures, covering more than 2.5 million unique word forms. The source code of these projects can be found on Github, while the binaries are published to jcenter and Maven Central, and the Docker images are published to Docker Hub.

DOI: 10.14750/ME.2022.020

8.2 Future work

There are several future development possibilities that have been identified during the development of the proposed models.

Two improvements that could be introduced relatively easily is to be able to inflect already inflected word forms, not just lemmas; and to be able to handle those affix type chains that contain the same affix type multiple times, e.g. in the Hungarian word *igaz**ság**os**ság*** (*~justice*). The first improvement would make Morpher more generic, while the second one would increase the number of word forms Morpher could handle.

To further optimize the Morpher model, artificial intelligence could be integrated to decrease the number of affix types to evaluate recursively during morphological analysis. For example, a neural network could be trained to provide some affix type candidates for the last affix type in a given word, and thus Morpher could drop the less probable affix type branches during analysis.

Another optimization option would be to use some kind of parallel processing platform. One such platform is Hadoop that I had already experimented with [6, 7], but maybe for this type of problem it would not be the best option, since the processing of affix types in an affix type chain is not independent from each other.

Although the current experiments have all been performed using the Hungarian language, the Morpher model could actually be extended to other languages as well. For this, new language handler implementations should be created, implementing the necessary API. For the Hungarian language, the Hunmorph-Ocamorph integration could be eliminated by making this layer more intelligent.

If there existed several language handler implementations, separate Morpher engine instances could be trained for the supported languages. These instances could be wrapped in a higher-level structure that we could call a Morpher system. Actually, in the Github repository, a simple Morpher system implementation is already implemented. It contains a a map of Morpher engines, each one associated with a language. During inflection generation and morphological analysis, the input not only contains the input word and affix types, but also the target language.

As an improvement, the Morpher system could be implemented in a way such that the internal Morpher engine instances are not totally isolated from each other. For example, the Morpher system could associate the valid lemmas of each language with concepts, connecting the words of the different engine instances. This way, the input could contain that the lemma to be inflected should be *alma*, the affix type should be plural, and the target language of the output should be English. The inflection response thus could contain the output word *apples*.

Morpher could also be extended to support languages that are not in the same language category as Hungarian. Languages that have a simpler morphology and use auxiliary words to modify the base meaning of their words, could also be supported. Morpher could return several words instead of a single inflected word form. As an example, if the input lemma is *table*, the required affix type is supressive case and the target language is English, then the output could be *on the table*.

Morpher is also capable of supporting higher-level grammatical models such as syntactic analyzers or free text processing frameworks. In the future, such higher-level models will be able to use Morpher to morphologically analyze input words, thus retrieving and incorporating morphological information of these words to solve higher-level grammatical problems.

DOI: 10.14750/ME.2022.020

Appendix A

Mapping of the Examined Annotation Token Systems

TABLE A.1: POS categories

Description	Ocamorph	Foma	Humor	Hunspell	Sample
Adjective	/ADJ	Adj	[MN]	adj	boldog *(happy)*
Adverb	/ADV	Adv	[HA]	adv	itt *(here)*
Article	/ART	-	-	det_indef	egy *(a/an)*
Conjunction	/CONJ	Con	[KOT]	con	vagy *(or)*
Determiner	/DET	Det	[DET]	det_def	ez *(this)*
Noun	/NOUN	Noun	[FN]	noun	alma *(apple)*
Numeral	/NUM	Num	[SZN]	adj_num	egy *(one)*
Onomatopoeic	/ONO	-	-	-	vau *(woof)*
Postposition	/POSTP	Post	[NU]	post	iránt *(towards)*
Preposition	/PREP	-	-	-	mint *(as)*
Utterance Interjection	/UTT-INT	Sentint	[ISZ]	sentint	kukucs *(peek-a-boo)*
Verb	/VERB	Verb	[IGE]	vrb	megy *(go)*

TABLE A.2: Noun features

Description	Ocamorph	Foma	Humor	Hunspell	Sample
Plural	⟨PLUR⟩	Plur	[PL]	PLUR	barátok *(friends)*
Plural familiar possessed	⟨PLUR⟨FAM⟩⟩	Fam	-	ék*	baráték *(group of friends)*
Possessor 1st person singular singular	⟨POSS⟨1⟩⟩	Posss1s	[PSe1]	POSS_ SG_1	barátom *(my friend)*
Continued on next page					

DOI: 10.14750/ME.2022.020

Table A.2 – continued from previous page

Description	Ocamorph	Foma	Humor	Hunspell	Sample
Possessor 2nd person singular singular	⟨POSS⟨2⟩⟩	Posss2s	[PSe2]	POSS_ SG_2	barátod *(your friend)*
Possessor 3rd person singular singular	⟨POSS⟩	Posss3s	[PSe3]	POSS_ SG_3	barátja *(her friend)*
Possessor 1st person plural singular	⟨POSS⟨1⟩⟨PLUR⟩⟩	Possp1s	[PSt1]	POSS_ PL_1	barátunk *(our friend)*
Possessor 2nd person plural singular	⟨POSS⟨2⟩⟨PLUR⟩⟩	Possp2s	[PSt2]	POSS_ PL_2	barátotok *(your friend)*
Possessor 3rd person plural singular	⟨POSS⟨PLUR⟩⟩	Possp3s	[PSt3]	POSS_ PL_3	barátuk *(their friend)*
Possessor 1st person singular plural	⟨PLUR⟩ ⟨POSS⟨1⟩⟩	Posss1p	[PSe1i]	-	barátaim *(my friends)*
Possessor 2nd person singular plural	⟨PLUR⟩ ⟨POSS⟨2⟩⟩	Posss2p	[PSe2i]	-	barátaid *(your friends)*
Possessor 3rd person singular plural	⟨PLUR⟩ ⟨POSS⟩	Posss3p	[PSe3i]	-	barátai *(her friends)*
Possessor 1st person plural plural	⟨PLUR⟩ ⟨POSS⟨1⟩⟨PLUR⟩⟩	Possp1p	[PSt1i]	-	barátaink *(our friends)*
Possessor 2nd person plural plural	⟨PLUR⟩ ⟨POSS⟨2⟩⟨PLUR⟩⟩	Possp2p	[PSt2i]	-	barátaitok *(your friends)*
Possessor 3rd person plural plural	⟨PLUR⟩ ⟨POSS⟨PLUR⟩⟩	Possp3p	[PSt3i]	-	barátaik *(their friends)*
Possessed singular	⟨ANP⟩	Gens	[POS]	POSS-ESSEE	baráté *(friend's)*
Possessed plural	⟨ANP⟨PLUR⟩⟩	Genpl	[POSi]	-	barátéi *(friend's things)*
Accusative	⟨CAS⟨ACC⟩⟩	Acc	[ACC]	ACC	almát *(the apple)*
Dative	⟨CAS⟨DAT⟩⟩	Dat	[DAT]	DAT	almának *(to the apple)*
Instrumental	⟨CAS⟨INS⟩⟩	Ins	[INS]	INSTR	almával *(with the apple)*

Continued on next page

DOI: 10.14750/ME.2022.020

Table A.2 – continued from previous page

Description	Ocamorph	Foma	Humor	Hunspell	Sample
Causative	⟨CAS⟨CAU⟩⟩	Cau	-	CAUS	almáért *(for the apple)*
Translative	⟨CAS⟨TRA⟩⟩	Fac	-	TRANS	almává *(into apple)*
Supressive	⟨CAS⟨SUE⟩⟩	Sup	[SUP]	SUE	almán *(on the apple)*
Sublative	⟨CAS⟨SBL⟩⟩	Sub	[SUB]	SBL	almára *(onto the apple)*
Delative	⟨CAS⟨DEL⟩⟩	Del	-	DEL	almáról *(from top of the apple)*
Inessive	⟨CAS⟨INE⟩⟩	Ine	[INE]	INE	almában *(in the apple)*
Elative	⟨CAS⟨ELA⟩⟩	Ela	-	ELA	almából *(from inside of the apple)*
Illative	⟨CAS⟨ILL⟩⟩	Ill	[ILL]	ILL	almába *(into the apple)*
Adessive	⟨CAS⟨ADE⟩⟩	Ade	-	ADE	almánál *(by the apple)*
Allative	⟨CAS⟨ALL⟩⟩	All	[ALL]	ALL	almához *(to the apple)*
Ablative	⟨CAS⟨ABL⟩⟩	Abl	-	ABL	almától *(from the apple)*
Temporal	⟨CAS⟨TEM⟩⟩	Tem	[TEM]	TEMP	hatkor *(at six)*
Terminative	⟨CAS⟨TER⟩⟩	Ter	[TER]	TERM	almáig *(to the apple)*
Formative	⟨CAS⟨FOR⟩⟩	For	-	FORM	almaként *(as an apple)*
Essive	⟨CAS⟨ESS⟩⟩	Ess	[ESS]	ESS	almául *(as an apple)*

TABLE A.3: Verb features

Description	Ocamorph	Foma	Humor	Hunspell	Sample (run)
Modal	⟨MODAL⟩	Pot	[_HAT]	hat*	futhat

Continued on next page

DOI: 10.14750/ME.2022.020

Table A.3 – continued from previous page

Description	Ocamorph	Foma	Humor	Hunspell	Sample (run)
Subjunctive Imperative 1st person singular indefinite	⟨SUBJUNC-IMP⟩ ⟨PERS⟨1⟩⟩	Conj Indef Sg1	[Pe1]	SUBJ/ IMPER_ INDEF_ SG_1	fussak
Subjunctive Imperative 2nd person singular indefinite	⟨SUBJUNC-IMP⟩ ⟨PERS⟨2⟩⟩	Conj Indef Sg2	[Pe2]	SUBJ/ IMPER_ INDEF_ SG_2	fuss
Subjunctive Imperative 3rd person singular indefinite	⟨SUBJUNC-IMP⟩	Conj Indef Sg3	[Pe3]	SUBJ/ IMPER_ INDEF_ SG_3	fusson
Subjunctive Imperative 1st person plural indefinite	⟨SUBJUNC-IMP⟩ ⟨PERS⟨1⟩⟩ ⟨PLUR⟩	Conj Indef Pl1	[Pt1]	SUBJ/ IMPER_ INDEF_ PL_1	fussunk
Subjunctive Imperative 2nd person plural indefinite	⟨SUBJUNC-IMP⟩ ⟨PERS⟨2⟩⟩ ⟨PLUR⟩	Conj Indef Pl2	[Pt2]	SUBJ/ IMPER_ INDEF_ PL_2	fussatok
Subjunctive Imperative 3rd person plural indefinite	⟨SUBJUNC-IMP⟩ ⟨PLUR⟩	Conj Indef Pl3	[Pt3]	SUBJ/ IMPER_ INDEF_ PL_3	fussanak
Subjunctive Imperative 1st person singular definite	⟨SUBJUNC-IMP⟩ ⟨PERS⟨1⟩⟩ ⟨DEF⟩	Conj Def Sg1	[TPe1]	SUBJ/ IMPER_ DEF_ SG_1	fussam
Subjunctive Imperative 1st person singular 2nd person object	⟨SUBJUNC-IMP⟩ ⟨PERS⟨1⟨OBJ⟨2⟩⟩⟩⟩	Conj Def Sg12	[IPe1]	SUBJ/ IMPER_ SG_1_ OBJ_2	fussalak
Subjunctive Imperative 2nd person singular definite	⟨SUBJUNC-IMP⟩ ⟨PERS⟨2⟩⟩ ⟨DEF⟩	Conj Def Sg2	[TPe2]	SUBJ/ IMPER_ DEF_ SG_2	fusd
Subjunctive Imperative 3rd person singular definite	⟨SUBJUNC-IMP⟩ ⟨DEF⟩	Conj Def Sg3	[TPe3]	SUBJ/ IMPER_ DEF_ SG_3	fussa
Subjunctive Imperative 1st person plural definite	⟨SUBJUNC-IMP⟩ ⟨PERS⟨1⟩⟩ ⟨PLUR⟩ ⟨DEF⟩	Conj Def Pl1	-	SUBJ/ IMPER_ DEF_ PL_1	fussuk

Continued on next page

DOI: 10.14750/ME.2022.020

Table A.3 – continued from previous page

Description	Ocamorph	Foma	Humor	Hunspell	Sample (run)
Subjunctive Imperative 2nd person plural definite	⟨SUBJUNC-IMP⟩ ⟨PERS⟨2⟩⟩ ⟨PLUR⟩ ⟨DEF⟩	Conj Def Pl2	-	SUBJ/ IMPER_ DEF_ PL_2	fussátok
Subjunctive Imperative 3rd person plural definite	⟨SUBJUNC-IMP⟩ ⟨PLUR⟩ ⟨DEF⟩	Conj Def Pl3	-	SUBJ/ IMPER_ DEF_ PL_3	fussák
Conditional 1st person singular indefinite	⟨COND⟩ ⟨PERS⟨1⟩⟩	Cond Indef Sg1	-	PRES_ COND_ INDEF_ SG_1	futnék
Conditional 2nd person singular indefinite	⟨COND⟩ ⟨PERS⟨2⟩⟩	Cond Indef Sg2	-	PRES_ COND_ INDEF_ SG_2	futnál
Conditional 3rd person singular indefinite	⟨COND⟩	Cond Indef Sg3	-	PRES_ COND_ INDEF_ SG_3	futna
Conditional 1st person plural indefinite	⟨COND⟩ ⟨PERS⟨1⟩⟩ ⟨PLUR⟩	Cond Indef Pl1	-	PRES_ COND_ INDEF_ PL_1	futnánk
Conditional 2nd person plural indefinite	⟨COND⟩ ⟨PERS⟨2⟩⟩ ⟨PLUR⟩	Cond Indef Pl2	-	PRES_ COND_ INDEF_ PL_2	futnátok
Conditional 3rd person plural indefinite	⟨COND⟩ ⟨PLUR⟩	Cond Indef Pl3	-	PRES_ COND_ INDEF_ PL_3	futnának
Conditional 1st person singular definite	⟨COND⟩ ⟨PERS⟨1⟩⟩ ⟨DEF⟩	Cond Def Sg1	-	PRES_ COND_ DEF_ SG_1	futnám
Conditional 1st person singular 2nd person object	⟨COND⟩ ⟨PERS⟨1⟨OBJ⟨2⟩⟩⟩⟩	Cond Def Sg12	-	PRES_ COND_ SG_1_ OBJ_2	futnálak
Conditional 2nd person singular definite	⟨COND⟩ ⟨PERS⟨2⟩⟩ ⟨DEF⟩	Cond Def Sg2	-	PRES_ COND_ DEF_ SG_2	futnád
Conditional 3rd person singular definite	⟨COND⟩ ⟨DEF⟩	Cond Def Sg3	-	PRES_ COND_ DEF_ SG_3	futná
Conditional 1st person plural definite	⟨COND⟩ ⟨PERS⟨1⟩⟩ ⟨PLUR⟩ ⟨DEF⟩	Cond Def Pl1	-	PRES_ COND_ DEF_ PL_1	futnánk

Continued on next page

DOI: 10.14750/ME.2022.020

Table A.3 – continued from previous page

Description	Ocamorph	Foma	Humor	Hunspell	Sample (run)
Conditional 2nd person plural definite	⟨COND⟩ ⟨PERS⟨2⟩⟩ ⟨PLUR⟩ ⟨DEF⟩	Cond Def Pl2	-	PRES_ COND_ DEF_ PL_2	futnátok
Conditional 3rd person plural definite	⟨COND⟩ ⟨PLUR⟩ ⟨DEF⟩	Cond Def Pl3	-	PRES_ COND_ DEF_ PL_3	futnák
Past 1st person singular indefinite	⟨PAST⟩ ⟨PERS⟨1⟩⟩	Past Indef Sg1	[Me1]	PAST_ INDIC_ INDEF_ SG_1	futottam
Past 2nd person singular indefinite	⟨PAST⟩ ⟨PERS⟨2⟩⟩	Past Indef Sg2	-	PAST_ INDIC_ INDEF_ SG_2	futottál
Past 3rd person singular indefinite	⟨PAST⟩	Past Indef Sg3	[Me3]	PAST_ INDIC_ INDEF_ SG_3	futott
Past 1st person plural indefinite	⟨PAST⟩ ⟨PERS⟨1⟩⟩ ⟨PLUR⟩	Past Indef Pl1	[Mt1]	PAST_ INDIC_ INDEF_ PL_1	futottunk
Past 2nd person plural indefinite	⟨PAST⟩ ⟨PERS⟨2⟩⟩ ⟨PLUR⟩	Past Indef Pl2	[Mt2]	PAST_ INDIC_ INDEF_ PL_2	futottatok
Past 3rd person plural indefinite	⟨PAST⟩ ⟨PLUR⟩	Past Indef Pl3	[Mt3]	PAST_ INDIC_ INDEF_ PL_3	futottak
Past 1st person singular definite	⟨PAST⟩ ⟨PERS⟨1⟩⟩ ⟨DEF⟩	Past Def Sg1	[TMe1]	PAST_ INDIC_ DEF_ SG_1	futottam
Past 2nd person singular definite	⟨PAST⟩ ⟨PERS⟨2⟩⟩ ⟨DEF⟩	Past Def Sg2	[TMe2]	PAST_ INDIC_ DEF_ SG_2	futottad
Past 1st person singular 2nd person object	⟨PAST⟩ ⟨PERS⟨1⟨OBJ⟨2⟩⟩⟩⟩	Past Def Sg12	[IMe1]	PAST_ INDIC_ SG_1_ OBJ_2	futottalak
Past 3rd person singular definite	⟨PAST⟩ ⟨DEF⟩	Past Def Sg3	[TMe3]	PAST_ INDIC_ DEF_ SG_3	futotta
Past 1st person plural definite	⟨PAST⟩ ⟨PERS⟨1⟩⟩ ⟨PLUR⟩ ⟨DEF⟩	Past Def Pl1	[TMt1]	PAST_ INDIC_ DEF_ PL_1	futottuk
Past 2nd person plural definite	⟨PAST⟩ ⟨PERS⟨2⟩⟩ ⟨PLUR⟩ ⟨DEF⟩	Past Def Pl2	-	PAST_ INDIC_ DEF_ PL_2	futottátok

Continued on next page

DOI: 10.14750/ME.2022.020

Table A.3 – continued from previous page

Description	Ocamorph	Foma	Humor	Hunspell	Sample (run)
Past 3rd person plural definite	⟨PAST⟩ ⟨PLUR⟩ ⟨DEF⟩	Past Def Pl3	-	PAST_ INDIC_ DEF_ PL_3	futották
1st person singular indefinite	⟨PERS⟨1⟩⟩	Indef Sg1	[e1]	SG_1	futok
1st person singular 2nd person object	⟨PERS⟨1⟨OBJ⟨2⟩⟩⟩⟩	Indef Sg12	[Ie1]	PRES_ INDIC_ SG_1_ OBJ_2	futlak
2nd person singular indefinite	⟨PERS⟨2⟩⟩	Indef Sg2	[e2]	SG_2	futsz
3rd person singular indefinite	⟨PERS⟩	Indef Sg3	[e3]	SG_3	fut
1st person plural indefinite	⟨PERS⟨1⟩⟩ ⟨PLUR⟩	Indef Pl1	[t1]	PL_1	futunk
2nd person plural indefinite	⟨PERS⟨2⟩⟩ ⟨PLUR⟩	Indef Pl2	[t2]	PL_2	futtok
3rd person plural indefinite	⟨VPLUR⟩	Indef Pl3	[t3]	PL_3	futnak
1st person singular definitive	⟨PERS⟨1⟩⟩ ⟨DEF⟩	Def Sg1	[Te1]	PRES_ INDIC_ DEF_ SG_1	futom
2nd person singular definitive	⟨PERS⟨2⟩⟩ ⟨DEF⟩	Def Sg2	[Te2]	PRES_ INDIC_ DEF_ SG_2	futod
3rd person singular definitive	⟨DEF⟩	Def Sg3	[Te3]	PRES_ INDIC_ DEF_ SG_3	futja
1st person plural definitive	⟨PERS⟨1⟩⟩ ⟨PLUR⟩ ⟨DEF⟩	Def Pl1	[Tt1]	PRES_ INDIC_ DEF_ PL_1	futjuk
2nd person plural definitive	⟨PERS⟨2⟩⟩ ⟨PLUR⟩ ⟨DEF⟩	Def Pl2	[Tt2]	PRES_ INDIC_ DEF_ PL_2	futjátok
3rd person plural definitive	⟨PLUR⟩ ⟨DEF⟩	Def Pl3	[Tt3]	PRES_ INDIC_ DEF_ PL_3	futják
Infinitive	⟨INF⟩	Inf	[INF]	ni*	futni *(to)*
Infinitive 1st person singular	⟨INF⟩ ⟨PERS⟨1⟩⟩	Inf11	[INRe1]	INF_ SG_1	futnom
Infinitive 2nd person singular	⟨INF⟩ ⟨PERS⟨2⟩⟩	Inf12	[INRe2]	INF_ SG_2	futnod

Continued on next page

DOI: 10.14750/ME.2022.020

Table A.3 – continued from previous page

Description	Ocamorph	Foma	Humor	Hunspell	Sample (run)
Infinitive 3rd person singular	⟨INF⟩ ⟨PERS⟩	Inf13	[INRe3]	INF_ SG_3	futnia
Infinitive 1st person plural	⟨INF⟩ ⟨PERS⟨1⟩⟩ ⟨PLUR⟩	Inf21	[INRt1]	INF_ PL_1	futnunk
Infinitive 2nd person plural	⟨INF⟩ ⟨PERS⟨2⟩⟩ ⟨PLUR⟩	Inf22	[INRt2]	INF_ PL_2	futnotok
Infinitive 3rd person plural	⟨INF⟩ ⟨PERS⟩ ⟨PLUR⟩	Inf23	[INRt3]	INF_ PL_3	futniuk

TABLE A.4: Noun derivations

Description	Ocamorph	Foma	Humor	Hunspell	Sample
Regular activity	[REG_ACT]	-	[_MIGY]	-	hülyéskedik *(fools around)*
Abstract	[ABSTRACT]	-	-	ság*	jóság *(goodness)*
Mrs	[MRS]	-	-	né*	királyné *(queen)*
Diminutive	[DIMIN]	-	[_DIM]	csk*	lányka *(girlie)*
Attributive	[ATTRIB]	-	[_SKEP]	s*	rituálés *(ritual)*
Metonymical attributive	[MET_ ATTRIB]	-	[_IKEP]	i*	politikai *(political)*
Metonymical attributive manner	[MET_ ATTRIB] [MANNER]	-	[_ILAG]	lag*	politikailag *(politically)*
Inalienable attributive	[INAL_ ATTRIB]	-	-	jú*	korú *(aging)*
Negative attributive	[NEG_ ATTRIB]	-	[_FFOSZT]	-	kedvtelen *(moody)*
Type 1	[TYPE1]	-	-	szerű*	újszerű *(novel)*
Type 2	[TYPE2]	-	-	féle*	jóféle *(good kind)*
Type 3	[TYPE3]	-	-	-	bárminemű *(anykind)*
Type 4	[TYPE4]	-	[_FAJTA]	fajta*	fajta *(breed)*
Type rank	[TYPE_ RANK]	-	-	-	másodrangú *(second-rate)*
Negative attributive 2	[NEG_ ATTRIB2]	-	-	mentes*	cukormentes *(sugar-free)*
Locative inessive	[LOC_INE]	-	[_BELI]	beli*	stílusbeli *(stylistic)*
Quantity	[QUANTITY]	-	[_MER]	nyi*	országnyi *(country-wise)*
Essivus formalis	[ESS_FOR]	Forp	-	képpen*	szükségképpen *(necessarily)*

Continued on next page

DOI: 10.14750/ME.2022.020

Table A.4 – continued from previous page

Description	Ocamorph	Foma	Humor	Hunspell	Sample
Comitative	[COM]	Soc	[_SOC]	stul*	kamatostul *(with interest)*
Period 1	[PERIOD1]	Dis	-	nként*	óránként *(per hour)*
Period 2	[PERIOD2]	-	[_NTA]	-	hetente *(weekly)*
Activity	[ACT]	-	[_FIT]	z*	tapétáz *(decorate)*
Activity 2	[ACT2]	-	[_FIL]	-	szomszédol *(visits the neighbors)*

TABLE A.5: Verb derivations

Description	Ocamorph	Foma	Humor	Hunspell	Sample
Frequentative	[FREQ]	-	-	nék*	futhatnék *(urge to run)*
Medial	[MEDIAL]	-	-	ódik*	íródik *(being written)*
Causative	[CAUS]	Imper	[_MUV]	tat*	csináltat *(have it done)*
Desiderative	[DESID]	Rep	[_GYAK]	gat*	írogat *(wishes to write)*
Adverbial participle	[PART]	-	[_HIN]	va*	írva *(written)*
Perfect adverbial participle	[PERF_ PART]	Advan	-	val*	írván *(while writing)*
Imperfect adjectival participle	[IMPERF_ PART]	-	-	ó*	író *(writer)*
Future adjectival participle	[FUT_ PART]	-	-	andó*	írandó *(to be written)*
Negative perfect adjectival participle	[NEG_ PERF_ PART]	-	[_IFOSZT]	talan*	íratlan *(unwritten)*
Gerund	[GERUND]	-	-	ás*	írás *(writing)*
Negative modal adjectival participle	[NEG_ MODAL_ PART]	-	[_HAT- ATLAN]	hatatlan_*	írhatatlan *(unwritable)*
Modal adjectival participle	[MODAL_ PART]	-	-	ható*	írható *(writable)*

DOI: 10.14750/ME.2022.020

TABLE A.6: Adjective derivations

Description	Ocamorph	Foma	Humor	Hunspell	Sample
Comparative	[COMPAR]	Mid	[_FOK]	bb*	jobb *(better)*
Superlative	[SUPERLAT]	-	[FF] [_FOK]	leg*	legjobb *(best)*
Supersuperlative	[SUPER SUPERLAT]	-	-	legesleg*	legesleg-jobb *(best of all)*
Comparative designative	[COMPAR_ DESIGN]	-	[_FOK] [_KIEM]	bbik*	jobbik *(better one)*
Superlative designative	[SUPERLAT_ DESIGN]	-	[FF] [_FOK] [_KIEM]	-	legjobbik *(best of them)*
Supersuperlative designative	[SUPER SUPERLAT_ DESIGN]	-	-	-	legesleg-jobbik *(best of them)*
Manner	[MANNER]	-	[_ESSMOD]	an*	egyszerűen *(easily)*
Intransitive resultative	[INTRANS_ RESULT]	-	[_MI]	odik*	erősödik *(getting stronger)*
Transitive resultative	[TRANS_ RESULT]	-	-	sít*	erősít *(streng-then)*

TABLE A.7: Numeral derivations

Description	Ocamorph	Foma	Humor	Hunspell	Sample
Multiplicative iterative	[MULTIPL-ITER]	Tmp	-	-	hatszor *(six times)*
Iterative attributive	[ITER_ ATTRIB]	Tmp Iadj	[_MUL]	-	egyszeri *(onefold)*
Multiplicative attributive	[MULTIPL_ ATTRIB]	Tmp Kas	-	szor*	egyszeres *(single)*
Multiplicative	[MULTIPL]	-	[_SZORTA]	szorta*	hatszorta *(six-time)*
Aggregative	[AGGREG]	-	[_ESSNUM]	-	ketten *(two of)*
Fractional	[FRACT]	Par	[_TORT]	d*	harmincad *(thirtieth)*
Ordinal	[ORD]	Par Ik	[_SORSZ]	dik*	harmadik *(third)*
Ordinal iterative	[ORD-ITER]	ParTmp	-	-	harmadszor *(thirdly)*
Ordinal iterative accomplished	[ORD-ITER-ACCOMPL]	Par Tmp Sub	-	-	harmadszorra *(for the third time)*
Date	[DATE]	-	[_DATUM]	dika*	hatodika *(sixth of)*

DOI: 10.14750/ME.2022.020

TABLE A.8: POSTP categories

Ocamorph	Foma	Humor	Hunspell	Sample
⟨POSTP⟨ALÁ⟩⟩	-	-	POSTP(alá)	ezalá *(below)*
⟨POSTP⟨ALATT⟩⟩	-	-	-	ezalatt *(under)*
⟨POSTP⟨ALÓL⟩⟩	-	-	-	ezalól *(from below)*
⟨POSTP⟨ÁLTAL⟩⟩	-	-	-	ezáltal *(by)*
⟨POSTP⟨ELÉ⟩⟩	-	-	POSTP(elé)	ezelé *(before)*
⟨POSTP⟨ELÉBE⟩⟩	-	-	-	ezelébe *(before)*
⟨POSTP⟨ELLEN⟩⟩	-	-	-	ezellen *(against)*
⟨POSTP⟨ELLENÉRE⟩⟩	-	-	-	ezellenére *(despite)*
⟨POSTP⟨ELŐL⟩⟩	-	-	-	ezelől *(from before)*
⟨POSTP⟨ELŐTT⟩⟩	-	-	-	ezelőtt *(before)*
⟨POSTP⟨FELETT⟩⟩	-	-	-	efelett *(above)*
⟨POSTP⟨FELÉ⟩⟩	-	-	-	efelé *(above)*
⟨POSTP⟨FELŐL⟩⟩	-	-	-	efelől *(from above)*
⟨POSTP⟨FELÜL⟩⟩	-	-	-	efelül *(furthermore)*
⟨POSTP⟨FÖLÉ⟩⟩	-	-	-	eföl é *(above)*
⟨POSTP⟨FÖLIBE⟩⟩	-	-	-	efölibe *(above)*
⟨POSTP⟨FÖLÜL⟩⟩	-	-	-	efölül *(above)*
⟨POSTP⟨HELYETT⟩⟩	-	-	-	ehelyett *(instead of)*
⟨POSTP⟨IRÁNT⟩⟩	-	-	-	ezirántt *(towards)*
⟨POSTP⟨KÖRÉ⟩⟩	-	-	-	eköré *(around)*
⟨POSTP⟨KÖRÖTT⟩⟩	-	-	-	ekörött *(around)*
⟨POSTP⟨KÖRÜL⟩⟩	-	-	-	körül *(around)*
⟨POSTP⟨KÖZBEN⟩⟩	-	-	-	eközben *(during)*
⟨POSTP⟨KÖZÉ⟩⟩	-	-	-	eközé *(among)*
⟨POSTP⟨KÖZIBE⟩⟩	-	-	-	eközibe *(among)*
⟨POSTP⟨KÖZÖTT⟩⟩	-	-	-	eközött *(among)*
⟨POSTP⟨KÖZÜL⟩⟩	-	-	-	eközül *(of)*
⟨POSTP⟨LÉTÉRE⟩⟩	-	-	-	létemre *(being)*

Continued on next page

DOI: 10.14750/ME.2022.020

Table A.8 – continued from previous page

Ocamorph	Foma	Humor	Hunspell	Sample
⟨POSTP⟨MELLETT⟩⟩	-	-	-	emellett *(besides)*
⟨POSTP⟨MELLÉ⟩⟩	-	-	-	emellé *(besides)*
⟨POSTP⟨MELLŐL⟩⟩	-	-	-	emellől *(from besides)*
⟨POSTP⟨MIATT⟩⟩	-	-	-	emiatt *(due to)*
⟨POSTP⟨MÖGÉ⟩⟩	-	-	-	emögé *(behind)*
⟨POSTP⟨MÖGÖTT⟩⟩	-	-	POSTP(mögött)	emögött *(behind)*
⟨POSTP⟨MÖGÜL⟩⟩	-	-	-	emögül *(from behind)*
⟨POSTP⟨NÉLKÜL⟩⟩	-	-	-	enélkül *(without)*
⟨POSTP⟨ÓTA⟩⟩	-	-	-	azóta *(since)*
⟨POSTP⟨RÉSZÉRE⟩⟩	-	-	-	részére *(for)*
⟨POSTP⟨RÉSZÉRŐL⟩⟩	-	-	-	részéről *(from your part)*
⟨POSTP⟨SZÁMÁRA⟩⟩	-	-	-	számára *(for)*
⟨POSTP⟨SZERINT⟩⟩	-	-	POSTP(szerint)	eszerint *(according to)*
⟨POSTP⟨UTÁN⟩⟩	-	-	-	ezután *(after)*
⟨POSTP⟨VÉGBŐL⟩⟩	-	-	-	evégből *(due to)*
⟨POSTP⟨VÉGETT⟩⟩	-	-	-	evégett *(due to)*
⟨POSTP⟨VÉGRE⟩⟩	-	-	-	evégre *(due to)*

DOI: 10.14750/ME.2022.020

Author's Publications

[1] Gábor Szabó and László Kovács. Benchmarking morphological analyzers for the Hungarian language. In *Annales Mathematicae et Informaticae*, volume 49, pages 141–166. Eszterházy Károly University Institute of Mathematics and Informatics, 2018. **Q3**. SJR: 0.157.

[2] G. Szabó and L. Kovács. Lattice based morphological rule induction. *Acta Universitatis Apulensis*, (53): 93–110, 2018.

[3] László Kovács and Gábor Szabó. String transformation based morphology learning. *Informatica*, 43 (4): 467–476, December 2019. **Q4**. SJR: 0.178.

[4] Gábor Szabó and László Kovács. Automated learning of hungarian morphology for inflection generation and morphological analysis. *Indonesian Journal of Electrical Engineering and Informatics (IJEEI)*, 8 (4): 746–756, December 2020. **Q4**. SJR: 0.168.

[5] Gábor Szabó and László Kovács. Optimization of the Morpher morphology engine using knowledge base reduction techniques. *Computing and Informatics*, 38 (4): 963–985, 2019. **Q3**. SJR: 0.217. Impact Factor: 0.524.

[6] László Kovács and Gábor Szabó. Utilizing Apache Hadoop in clique detection methods. *Production Systems and Information Engineering*, 7: 43–53, 2015.

[7] László Kovács and Gábor Szabó. Conceptualization with incremental Bron-Kerbosch algorithm in big data architecture. *Acta Polytechnica Hungarica*, 13 (2), 2016. **Q2**. SJR: 0.298. Impact Factor: 1.219. Independent citations: 7.

[8] Gábor Szabó. Efficient method for training set generation in morphological analysis. In *Doktoranduszok Fóruma - Gépészmérnöki és Informatikai Kar szekciókiadványa*, November 2014.

[9] G. Szabó and L. Kovács. Efficiency analysis of inflection rule induction. In *Proceedings of the 2015 16th International Carpathian Control Conference (ICCC)*, pages 521–525, May 2015.

[10] Gábor Szabó. Grammatical rule generation strategies for concept lattice based inflection systems. In *Doktoranduszok Fóruma - Gépészmérnöki és Informatikai Kar szekciókiadványa*, pages 5–10, November 2015.

[11] László Kovács and Gábor Szabó. String transformation approach for morpheme rule induction. *Procedia Technology*, 22: 854–861, 2016.

[12] Gábor Szabó. Edit distance based grammatical rule generation using an improved cost function. In *The Publications of the MultiScience - XXX. microCAD International Multidisciplinary Scientific Conference*, pages 1–8, University of Miskolc, Hungary, April 2016.

DOI: 10.14750/ME.2022.020

[13] Gábor Szabó. Lattice-based ruleset representation for morpheme analysis. In *Doktoranduszok Fóruma - Gépészmérnöki és Informatikai Kar szekciókiadványa*, November 2016.

[14] László Kovács and Szabó Gábor. Generalization of string transformation rules using optimized concept lattice construction method. *Procedia Engineering*, 181: 604–611, 2017. SJR: 0.286. Independent citations: 1.

[15] Gábor Szabó. Morphological models for natural languages. In *XX. Tavaszi Szél Konferencia 2017*, March 2017.

[16] Gábor Szabó. Computational models for morphology. In *The Publications of the MultiScience - XXXI. microCAD International Multidisciplinary Scientific Conference*, pages 1–8, University of Miskolc, Hungary, April 2017.

DOI: 10.14750/ME.2022.020

References

[Ahlberg et al., 2015] Ahlberg, M., Forsberg, M., and Hulden, M. (2015). Paradigm classification in supervised learning of morphology. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1024–1029.

[Aisha and Sun, 2009] Aisha, B. and Sun, M. (2009). A Uyghur morpheme analysis method based on conditional random fields. *Int. J. of Asian Lang. Proc.*, 19:69–84.

[Akmajian et al., 2017] Akmajian, A., Farmer, A. K., Bickmore, L., Demers, R. A., and Harnish, R. M. (2017). *Linguistics: An introduction to language and communication*. MIT press.

[Barton, 1986] Barton, G. E. (1986). Computational complexity in two-level morphology. In *Proceedings of the 24th annual meeting on Association for Computational Linguistics*, pages 53–59. Association for Computational Linguistics.

[Bauer, 2003] Bauer, L. (2003). Introducing linguistic morphology.

[Bergmanis and Goldwater, 2017] Bergmanis, T. and Goldwater, S. (2017). From segmentation to analyses: A probabilistic model for unsupervised morphology induction. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 337–346. Association for Computational Linguistics.

[Birkhoff, 1940] Birkhoff, G. (1940). *Lattice theory*, volume 25. American Mathematical Soc.

[Booij, 2012] Booij, G. (2012). *The grammar of words: An introduction to linguistic morphology*. Oxford University Press.

[Chan and Yang, 2008] Chan, E. and Yang, C. D. (2008). Structures and distributions in morphology learning. *University of Pennsylvania, Philadelphia, PA*.

[Clark, 2001] Clark, A. (2001). Learning morphology with pair hidden Markov models. In *ACL (Companion Volume)*, pages 55–60. Citeseer.

[Cotterell et al., 2018] Cotterell, R., Kirov, C., Sylak-Glassman, J., Walther, G., Vylomova, E., McCarthy, A. D., Kann, K., Mielke, S., Nicolai, G., Silfverberg, M., Yarowsky, D., Eisner, J., and Hulden, M. (2018). The CoNLL–SIGMORPHON 2018 shared task: Universal morphological reinflection. In *Proceedings of the CoNLL–SIGMORPHON 2018 Shared Task: Universal Morphological Reinflection*, pages 1–27, Brussels. Association for Computational Linguistics.

[Cotterell et al., 2017] Cotterell, R., Kirov, C., Sylak-Glassman, J., Walther, G., Vylomova, E., Xia, P., Faruqui, M., Kübler, S., Yarowsky, D., Eisner, J., and Hulden, M. (2017). CoNLL-SIGMORPHON 2017 shared task: Universal morphological reinflection in 52 languages. In *Proceedings of the CoNLL SIGMORPHON 2017 Shared*

DOI: 10.14750/ME.2022.020

Task: Universal Morphological Reinflection, pages 1–30, Vancouver. Association for Computational Linguistics.

[Cotterell et al., 2016] Cotterell, R., Kirov, C., Sylak-Glassman, J., Yarowsky, D., Eisner, J., and Hulden, M. (2016). The SIGMORPHON 2016 shared task — Morphological reinflection. In *Proceedings of the 2016 Meeting of SIGMORPHON*, Berlin, Germany. Association for Computational Linguistics.

[Cotterell et al., 2015] Cotterell, R., Müller, T., Fraser, A., and Schütze, H. (2015). Labeled morphological segmentation with semi-Markov models. In *Proceedings of the Nineteenth Conference on Computational Natural Language Learning*, pages 164–174.

[Creutz and Lagus, 2002] Creutz, M. and Lagus, K. (2002). Unsupervised discovery of morphemes. In *Proceedings of the ACL-02 workshop on Morphological and phonological learning-Volume 6*, pages 21–30. Association for Computational Linguistics.

[Creutz and Lagus, 2004] Creutz, M. and Lagus, K. (2004). Induction of a simple morphology for highly-inflecting languages. In *Proceedings of the 7th Meeting of the ACL Special Interest Group in Computational Phonology: Current Themes in Computational Phonology and Morphology*, SIGMorPhon '04, pages 43–51, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Creutz and Lagus, 2005a] Creutz, M. and Lagus, K. (2005a). Inducing the morphological lexicon of a natural language from unannotated text. In *Proceedings of the International and Interdisciplinary Conference on Adaptive Knowledge Representation and Reasoning (AKRR'05)*, volume 1, pages 51–59.

[Creutz and Lagus, 2005b] Creutz, M. and Lagus, K. (2005b). *Unsupervised morpheme segmentation and morphology induction from text corpora using Morfessor 1.0*. Helsinki University of Technology Helsinki.

[Csendes et al., 2004] Csendes, D., Csirik, J., and Gyimóthy, T. (2004). The Szeged Corpus: A POS tagged and syntactically annotated Hungarian natural language corpus. In *International Conference on Text, Speech and Dialogue*, pages 41–47. Springer.

[De la Higuera, 2010] De la Higuera, C. (2010). *Grammatical inference: Learning automata and grammars*. Cambridge University Press.

[Endrédy, 2015] Endrédy, I. (2015). Corpus based evaluation of stemmers. In Vetulani, Z. and Mariani, J., editors, *7th Language & Technology Conference: Human Language Technologies as a Challenge for Computer Science and Linguistics*. Poznań: Uniwersytet im. Adama Mickiewicza w Poznaniu, Poznań: Uniwersytet im. Adama Mickiewicza w Poznaniu.

[Endrédy and Novák, 2015] Endrédy, I. and Novák, A. (2015). Szótövesítő programok összehasonlítása és alkalmazásaik. *Alkalmazott Nyelvtudomány*, 15:7–27.

[Faruqui et al., 2015] Faruqui, M., Tsvetkov, Y., Neubig, G., and Dyer, C. (2015). Morphological inflection generation using character sequence to sequence learning. *arXiv preprint arXiv:1512.06110*.

[Ganter and Wille, 2012] Ganter, B. and Wille, R. (2012). *Formal concept analysis: Mathematical foundations*. Springer Science & Business Media.

DOI: 10.14750/ME.2022.020

[Gelbukh et al., 2004] Gelbukh, A., Alexandrov, M., and Han, S.-Y. (2004). Detecting inflection patterns in natural language by minimization of morphological model. In *Iberoamerican Congress on Pattern Recognition*, pages 432–438. Springer.

[Gelbukh and Sidorov, 2003] Gelbukh, A. and Sidorov, G. (2003). Approach to construction of automatic morphological analysis systems for inflective languages with little effort. volume 2588, pages 215–220.

[Goldsmith, 2001] Goldsmith, J. (2001). Unsupervised learning of the morphology of a natural language. *Computational Linguistics*, 27(2):153–198.

[Goldsmith, 2006] Goldsmith, J. (2006). An algorithm for the unsupervised learning of morphology. *Natural language engineering*, 12(4):353–371.

[Goldwater and Johnson, 2004] Goldwater, S. and Johnson, M. (2004). Priors in Bayesian learning of phonological rules. In *Proceedings of the 7th Meeting of the ACL Special Interest Group in Computational Phonology: Current Themes in Computational Phonology and Morphology*, pages 35–42. Association for Computational Linguistics.

[Grätzer, 2003] Grätzer, G. (2003). *General Lattice Theory*. Birkhäuser Verlag.

[Grönroos et al., 2014] Grönroos, S.-A., Virpioja, S., Smit, P., and Kurimo, M. (2014). Morfessor FlatCat: An HMM-based method for unsupervised and semi-supervised learning of morphology. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 1177–1185.

[Gruber, 1993] Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge acquisition*, 5(2):199–220.

[Hajič, 1988] Hajič, J. (1988). Formal morphology. In *Proceedings of the 12th Conference on Computational Linguistics - Volume 1*, COLING '88, pages 222–224, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Halácsy et al., 2003] Halácsy, P., Kornai, A., Németh, L., Rung, A., and Szakadát, I. (2003). A Szószablya projekt. In Alexin, Z. and Csendes, D., editors, *I. Magyar Számítógépes Nyelvészeti Konferencia előadásai*, pages 298–299.

[Hulden, 2009] Hulden, M. (2009). Foma: A finite-state compiler and library. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics: Demonstrations Session*, pages 29–32. Association for Computational Linguistics.

[Hulden, 2014] Hulden, M. (2014). Generalizing inflection tables into paradigms with finite state operations. In *Proceedings of the 2014 Joint Meeting of SIGMORPHON and SIGFSM*, pages 29–36.

[Kann and Schütze, 2017] Kann, K. and Schütze, H. (2017). Unlabeled data for morphological generation with character-based sequence-to-sequence models. In *Proceedings of the First Workshop on Subword and Character Level Models in NLP*, pages 76–81, Copenhagen, Denmark. Association for Computational Linguistics.

[Kohonen et al., 2010] Kohonen, O., Virpioja, S., and Lagus, K. (2010). Semi-supervised learning of concatenative morphology. In *Proceedings of the 11th Meeting of the ACL Special Interest Group on Computational Morphology and Phonology*, pages 78–86. Association for Computational Linguistics.

DOI: 10.14750/ME.2022.020

[Koskenniemi, 1983] Koskenniemi, K. (1983). *Two-level morphology: A general computational model for word-form recognition and production*, volume 11. University of Helsinki, Department of General Linguistics Helsinki.

[Lafferty et al., 2001] Lafferty, J., McCallum, A., and Pereira, F. C. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data.

[Lee and Goldsmith, 2016] Lee, J. L. and Goldsmith, J. A. (2016). Linguistica 5: Unsupervised learning of linguistic structure. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics*, pages 22–26, San Diego, California. Association for Computational Linguistics.

[Levenshtein, 1966] Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10:707.

[Lignos, 2010] Lignos, C. (2010). Learning from unseen data. In Kurimo, M., Virpioja, S., and Turunen, V. T., editors, *Proceedings of the Morpho Challenge 2010 Workshop*, pages 35–38, Helsinki, Finland. Aalto University School of Science and Technology.

[Lignos et al., 2009] Lignos, C., Chan, E., Marcus, M. P., and Yang, C. (2009). A rule-based unsupervised morphology learning framework. In *CLEF (Working Notes)*.

[Luong et al., 2013] Luong, T., Socher, R., and Manning, C. (2013). Better word representations with recursive neural networks for morphology. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 104–113.

[McCandless et al., 2010] McCandless, M., Hatcher, E., Gospodnetić, O., and Gospodnetić, O. (2010). *Lucene in action*, volume 2. Manning Greenwich.

[McCarthy et al., 2019] McCarthy, A. D., Vylomova, E., Wu, S., Malaviya, C., Wolf-Sonkin, L., Nicolai, G., Kirov, C., Silfverberg, M., Mielke, S. J., Heinz, J., Cotterell, R., and Hulden, M. (2019). The SIGMORPHON 2019 shared task: Morphological analysis in context and cross-lingual transfer for inflection. In *Proceedings of the 16th Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 229–244, Florence, Italy. Association for Computational Linguistics.

[Meyer, 2009] Meyer, C. F. (2009). *Introducing English linguistics*. Cambridge University Press.

[Miháltz et al., 2008] Miháltz, M., Hatvani, C., Kuti, J., Szarvas, G., Csirik, J., Prószéky, G., and Váradi, T. (2008). Methods and results of the Hungarian WordNet project. In *Proceedings of the Fourth Global WordNet Conference. GWC*, pages 387–405.

[Miller, 1998] Miller, G. (1998). *WordNet: An electronic lexical database*. MIT press.

[Mohri, 1997] Mohri, M. (1997). Finite-state transducers in language and speech processing. *Computational linguistics*, 23(2):269–311.

[Müller et al., 2015] Müller, T., Cotterell, R., Fraser, A., and Schütze, H. (2015). Joint lemmatization and morphological tagging with lemming. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2268–2274.

DOI: 10.14750/ME.2022.020

[Müller and Schütze, 2015] Müller, T. and Schütze, H. (2015). Robust morphological tagging with word representations. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 526–536, Denver, Colorado. Association for Computational Linguistics.

[Narasimhan et al., 2015] Narasimhan, K., Barzilay, R., and Jaakkola, T. (2015). An unsupervised method for uncovering morphological chains. *Transactions of the Association for Computational Linguistics*, 3:157–167.

[Oncina, 1998] Oncina, J. (1998). The data driven approach applied to the OSTIA algorithm. In *International Colloquium on Grammatical Inference*, pages 50–56. Springer.

[Oncina et al., 1993] Oncina, J., García, P., and Vidal, E. (1993). Learning subsequential transducers for pattern recognition interpretation tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(5):448–458.

[Ore, 1944] Ore, O. (1944). Galois connexions. *Transactions of the American Mathematical Society*, 55(3):493–513.

[Östling, 2016] Östling, R. (2016). Morphological reinflection with convolutional neural networks. In *Proceedings of the 14th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 23–26, Berlin, Germany. Association for Computational Linguistics.

[Pirinen et al., 2010] Pirinen, T., Lindén, K., et al. (2010). Creating and weighting hunspell dictionaries as finite-state automata. *Investigationes Linguisticae (Online Edition)*.

[Poon et al., 2009] Poon, H., Cherry, C., and Toutanova, K. (2009). Unsupervised morphological segmentation with log-linear models. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 209–217. Association for Computational Linguistics.

[Porter, 1980] Porter, M. F. (1980). An algorithm for suffix stripping. *Program*, 14(3):130–137.

[Porter, 2001] Porter, M. F. (2001). Snowball: A language for stemming algorithms.

[Prószéky and Kis, 1999] Prószéky, G. and Kis, B. (1999). A unification-based approach to morpho-syntactic parsing of agglutinative and other (highly) inflectional languages. In *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*, pages 261–268. Association for Computational Linguistics.

[Prószéky and Novák, 2005] Prószéky, G. and Novák, A. (2005). Computational morphologies for small Uralic languages. In *Inquiries into Words, Constraints and Contexts Festschrift in the Honour of Kimmo Koskenniemi on his 60th Birthday*, pages 116–125. Gummerus Printing, Saarijärvi/CSLI Publications, Stanford.

[Prószéky and Tihanyi, 1993] Prószéky, G. and Tihanyi, L. (1993). Humor: High-speed unification morphology and its applications for agglutinative languages. *La tribune des industries de la langue*, 10:28–29.

DOI: 10.14750/ME.2022.020

[Ruokolainen et al., 2016] Ruokolainen, T., Kohonen, O., Sirts, K., Grönroos, S.-A., Kurimo, M., and Virpioja, S. (2016). A comparative study of minimally supervised morphological segmentation. *Computational Linguistics*, 42(1):91–120.

[Ruokolainen et al., 2014] Ruokolainen, T., Kohonen, O., Virpioja, S., et al. (2014). Painless semi-supervised morphological segmentation using conditional random fields. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics, volume 2: Short Papers*, pages 84–89.

[Satta and Henderson, 1997] Satta, G. and Henderson, J. C. (1997). String transformation learning. In *Proceedings of the eighth conference on European chapter of the Association for Computational Linguistics*, pages 444–451. Association for Computational Linguistics.

[Schröder et al., 2018] Schröder, F., Kamlot, M., Billing, G., and Köhn, A. (2018). Finding the way from ä to a: Sub-character morphological inflection for the SIGMORPHON 2018 shared task. In *Proceedings of the CoNLL–SIGMORPHON 2018 Shared Task: Universal Morphological Reinflection*, pages 76–85, Brussels. Association for Computational Linguistics.

[Senuma and Aizawa, 2017] Senuma, H. and Aizawa, A. (2017). Seq2seq for morphological reinflection: When deep learning fails. In *Proceedings of the CoNLL SIGMORPHON 2017 Shared Task: Universal Morphological Reinflection*, pages 100–109, Vancouver. Association for Computational Linguistics.

[Shalonova and Flach, 2007] Shalonova, K. and Flach, P. (2007). Morphology learning using tree of aligned suffix rules. In *ICML Workshop: Challenges and Applications of Grammar Induction*.

[Sharma et al., 2018] Sharma, A., Katrapati, G., and Sharma, D. M. (2018). IIT(BHU)–IIITH at CoNLL–SIGMORPHON 2018 shared task on universal morphological reinflection. In *Proceedings of the CoNLL–SIGMORPHON 2018 Shared Task: Universal Morphological Reinflection*, pages 105–111, Brussels. Association for Computational Linguistics.

[Snover and Brent, 2003] Snover, M. G. and Brent, M. R. (2003). A probabilistic model for learning concatenative morphology. In *Advances in Neural Information Processing Systems*, pages 1537–1544.

[Soricut and Och, 2015] Soricut, R. and Och, F. (2015). Unsupervised morphology induction using word embeddings. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1627–1637.

[Sorokin, 2018] Sorokin, A. (2018). What can we gain from language models for morphological inflection? In *Proceedings of the CoNLL–SIGMORPHON 2018 Shared Task: Universal Morphological Reinflection*, pages 99–104, Brussels. Association for Computational Linguistics.

[Tepper and Xia, 2010] Tepper, M. and Xia, F. (2010). Inducing morphemes using light knowledge. *ACM Transactions on Asian Language Information Processing (TALIP)*, 9(1):3.

[Theron and Cloete, 1997] Theron, P. and Cloete, I. (1997). Automatic acquisition of two-level morphological rules. In *Proceedings of the fifth conference on Applied natural language processing*, pages 103–110. Association for Computational Linguistics.

DOI: 10.14750/ME.2022.020

[Tordai and de Rijke, 2006] Tordai, A. and de Rijke, M. (2006). *Four stemmers and a funeral: Stemming in Hungarian at CLEF 2005*, pages 179–186. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Tóth and Kovács, 2014] Tóth, Z. and Kovács, L. (2014). Induction of inflection rules with classification and associative memory for Hungarian language. *Scientific Bulletin of the Petru Maior University of Targu Mures*, 11(2).

[Trón et al., 2006] Trón, V., Halácsy, P., Rebrus, P., Rung, A., Vajda, P., and Simon, E. (2006). Morphdb.hu: Hungarian lexical database and morphological grammar. In *LREC*, pages 1670–1673. Citeseer.

[Trón et al., 2005] Trón, V., Kornai, A., Gyepesi, G., Németh, L., Halácsy, P., and Varga, D. (2005). Hunmorph: Open source word analysis. In *Proceedings of the Workshop on Software*, pages 77–85. Association for Computational Linguistics.

[Virpioja et al., 2013] Virpioja, S., Smit, P., Grönroos, S.-A., and Kurimo, M. (2013). Morfessor 2.0: Python implementation and extensions for Morfessor Baseline. Technical report.

[Zhao and Yao, 2006] Zhao, Y. and Yao, Y. (2006). Classification based on logical concept analysis. In *Conference of the Canadian Society for Computational Studies of Intelligence*, pages 419–430. Springer.

[Zhu et al., 2017] Zhu, Q., Li, Y., and Li, X. (2017). Character sequence-to-sequence model with global attention for universal morphological reinflection. In *Proceedings of the CoNLL SIGMORPHON 2017 Shared Task: Universal Morphological Reinflection*, pages 85–89, Vancouver. Association for Computational Linguistics.