Charles University in Prague

Faculty of Mathematics and Physics

MASTER'S THESIS

Radoslav Klíč

Acquisition of inflectional paradigms with minimal supervision

Institute of Formal and Applied Linguistics

Supervisor of the master's thesis:	RNDr. Jiří Hana Ph.D.
Study programme:	Computer Science
Specialisation:	Computational and Formal Linguistics

Prague 2012

I would like to thank my supervisor, doctor Hana, for professional guidance, a lot of useful advice and setting an example of a professional scientist to me.

Název práce: Automatické osvojení vzorů s minimální supervizí

Autor: Radoslav Klíč

Katedra: Ústav formální a aplikované lingvistiky (ÚFAL)

Vedoucí diplomové práce: RNDr. Jiří Hana Ph.D., ÚFAL

Abstrakt: Diplomová práce popisuje algoritmus pro automatické osvojení vzorů s minimální supervizí, který vznikl rozšířením systému Paramor (Monson, 2009), fungujícího zcela bez supervize. Systém je modifikován, aby přijímal snadno dostupná data ve formě ohýbaných slov s označenou hranicí morfémů jako dodatečný vstup. Součástí práce je také knihovna pro hierarchické shlukování, která umožňuje kombinaci různých zdrojů informací. Přístup byl testován na češtině, slovinštině, němčině a katalánštině a vykázal zvýšenou F-míru v porovnáni se základním Paramorem.

Klíčová slova: strojové učení, morfologie, fonologie, vzory ohýbaní slov

Title: Acquisition of inflectional paradigms with minimal supervision

Author: Radoslav Klíč

Department: Institute of Formal and Applied Linguistics

Supervisor: RNDr. Jiří Hana Ph.D., Institute of Formal and Applied Linguistics

Abstract: The thesis presents a semi-supervised morphology learner developed by extending Paramor (Monson, 2009), an unsupervised system, to accept easy to obtain manually provided data in the form of inflections with marked morpheme boundary. In addition, a hierarchical clustering framework allowing combination of multiple sources of information was developed as a part of the thesis. The approach was tested on Czech, Slovene, German and Catalan and has shown increased F-measure in comparison with the Paramor baseline.

Keywords: machine learning, morphology, phonology, inflectional paradigms

Contents

Chapter 1

Introduction

Morphological analysis is important for many applications working with natural language in written or spoken form. An analyser can be built manually (for example, using finite-state technology) or machine learning (ML) with various levels of supervision can be employed. Completely supervised techniques and hand-written analysers can achieve very high quality of results.

However, supervised ML approaches usually require an annotated corpus, which for many of the world's languages is an unavailable resource and it would be expensive or straight out impossible to create one. Similar problem arises with the hand-written analysers, for which experts with deep knowledge of given language and strong linguistic and programming background are needed. Another important resource to be taken into account is time, as creating an analyser manually by experts is considerably time-consuming.

In cases where the expensive resources are not available, one may use one of the unsupervised or semi-supervised morphology learners. The semi-supervised approaches are attempting to use the fact that even for underresourced languages there is (nearly) always possible to get some useful resources without spending much time or money. These resources may include inflections of individual words or inflection classes from a grammar textbook.

Analyses vary not only in the type and amount of learning data they require, but also in the output they provide to the user. Morphological analysis in its standard sense returns all possible lemmas and part-of-speech tags for each word in the text, disregarding the context. For example, *handles* should be analysed as a 3$^{\text{rd}}$ person singular verb form and a plural noun; using the Penn Treebank tagset (Marcus et al., 1993) this could be encoded as $\{$(*handle*, VBZ), (*handle*, NNS)$\}$. Resource-light approaches to MA are not always able to provide such detailed analysis, but can produce results such as:

1. Related word forms. For *run*, the analyser may return *runs*, *running*. This kind

of analysis is useful in information retrieval. For example, if you search for *table* on Google, you expect it to find *tables* as well.

2. Morphemic segmentation, i.e. splitting words into morphemes. For *tables*, *table* + *s* is returned. Speech recognition systems can benefit from this kind of analysis by building morpheme level language models.

The system developed for this thesis is semi-supervised as it uses manually entered inflections with marked morpheme boundary as one of its inputs. It contains two almost independent modules:

1. Modified Paramor (Monson, 2009) analyser. Modifications aim at improving its results by using the manual seed mentioned above. The seed is used (1) to help Paramor recognise true suffixes and paradigms and (2) generate simple rules capturing stem allomorphy. Treating allomorfic variants as a single stem enables the algorithm to find more complete paradigms and place morpheme boundaries more adequately. Like the original Paramor, the system is capable of inducing inflectional paradigms as well as using them for morphemic segmentation.

2. Word clustering framework which allows combination of multiple sources of information, for example output of a paradigm producing algorithm such as the one from (1), morphology segmentation algorithm or modified edit distance. It also provides evaluation of resulting clusters' adequacy if a lemmatised corpus is available.

The target languages are inflectional, especially Slavic languages, but the system should yield acceptable results for any language with suffixal morphology.

1.1 Terminology

Terms from the field of morphology are used quite loosely in various publications. In this section, I briefly summarise some of them and clarify in what sense they will be used in this thesis.

Morpheme is the smallest unit in a language bearing a meaning. It is an abstract entity, which may be realised in speech or writing by more than one form. The individual realisations are called **morphs**. Every word consist of one or more morphs e.g., *talk + ing, un + avail + able*.

Allomorphs are morphs realising the same morpheme, depending on the context. They occur due to phonological/graphemic changes or irregularities. For example, consider the declension of the Czech word *matka* 'mother' in Table 1.1. It exhibits stem-final consonant change (palatalisation of *k* to *c*) triggered by the dative and local singular ending, and epenthesis (insertion of *-e-*) in the bare stem genitive plural. As a result, the stem has 3 allomorphs: *matk*, *matc* and *matek*.

Case	**Singular**	**Plural**
nom	mat**k**+a	mat**k**+y
gen	mat**k**+y	mat**ek**+0
dat	mat**c**+e	mat**k**+ám
acc	mat**k**+u	mat**k**+y
voc	mat**k**+o	mat**k**+y
loc	mat**c**+e	mat**k**+ách
inst	mat**k**+ou	mat**k**+ami

Table 1.1: Declension of the word *matka* 'mother'. Changing part of the stem is in bold.

Inflection is a morphological process which creates different forms of one word:

walk, walks, walked, walking

Derivation is a process which creates new words:

relent $\rightarrow$ *relentless* $\rightarrow$ *relentlessness*

Lexeme is a set of words related by inflection. For example, all forms of the word *matka* 'mother' in Table 1.1 form a lexeme. Lexeme is also sometimes used in a related but slightly different meaning – as a unit of language that has a semantic meaning.

Canonical form, or dictionary form, is a form selected by a convention to represent a lexeme. For example, usually infinitive is chosen for verbs and nominative singular for nouns.

Lemma is a unique identifier for a lexeme. Typically, the canonical form can serve as a lemma (possibly with indexes disambiguating polysemous words, such as $bank_1$/$bank_2$). Nevertheless, lemma can be any arbitrarily chosen unique identifier (at least in NLP).

Inflections of most of the words follow regular patterns, called **paradigms**. Definitions of a paradigm in various works differ. Paradigms as taught in Czech schools consist of suffix for each combination of relevant grammatical categories[1] and a word inflected according to the paradigm. This word serves as an example and an easy-to-remember identifier of the paradigm. In this thesis, the term paradigm will be used for a set of suffixes.

[1]case and number for nouns, person and number for verbs

1.2 Layout of the thesis

The rest of the thesis is organised as follows: Chapter 2 reviews some of the important previous works related to the subject of the thesis. Detailed description of Paramor is provided in Chapter 3. Chapter 4 presents modifications of Paramor to handle basic allomorphy and use a manually provided seed. The clustering framework and implemented string distance measures are described in Chapter 5. Experiments and their results are presented in Chapter 6. Chapter 7 summarises the thesis and outlines possible future improvements.

Chapter 2

Previous Works

The area of computational morphology is wide and rich with publications. Approaches to computational morphology can be roughly classified into 3 basic categories:

1. Unsupervised morphology learners: algorithms which use solely plain-text corpora or word-lists as their input.

2. Supervised systems and hand-built systems. They require annotated corpora or manually created rules together with large vocabularies.

3. Semi-supervised and resource-light approaches. The individual approaches vary in the amount and form of human supervision. The forms of human supervision may include providing data such as inflected word forms, paradigms, affix lists, phonological rules etc. Human supervisor may also be employed in an iterative training process by being asked to interact in each iteration of the learning process.

In this chapter, some of the existing systems are described, with main focus on the unsupervised and semi-supervised methods.

2.1 Creating analysers by experts

The two most commonly used approaches to creating hand-made morphology analysers are 2-level morphology (Koskenniemi, 1983; Karttunen et al., 2003) and the so-called "engineering approach".

2.1.1 Two-level morphology (2LM)

2LM is a popular formalism, which works with two levels of representation of words: an abstract *lexical level* and a concrete *surface level*, for example:

```
b a b y + 0 s
b a b i 0 e s
```

with an inline notation

```
b:b a:a b:b y:i +:0 0:e s:s
```

relates the lexical (upper) and surface (lower) form of the word *babies* . The + character denotes the morpheme boundary, 0 is an empty string (there has to be a 1:1 correspondence between characters on each level). The two levels are related by declarative context-sensitive rules such as:

```
y:i <= _ +:0 s:s
0:e <= y:i +:0 _ s:s
```

The rules above describe changing of *y* to *i* before the morpheme boundary followed by *s* and insertion of *e* between *i* and *s* (in the specified context). The underscore sign denotes the position where the change is taking place.

The designer of the analyser specifies the rules and a network of lexicons containing morphemes in their lexical form. The network also specifies basic morphotactics of morphemes in the lexicons. Rules and lexicons are then compiled into a finite-state transducer. The formalism is powerful enough to express most of the phonological and morphological processes, although their description by the rules is not always straightforward.

2.1.2 Engineering approach

The engineering approach, represented by (Hajič, 2004), models morphology as a purely concatenative process. Changing parts of the stem are moved to the suffixes and as a result, high number of paradigms emerge. For example, to model the English past tense one would need to create paradims as (*0, ed*), (*0, ped*), (*0, ted*), (*y, ied*) etc. Table 2.1 shows how would an “engineering” paradigm for the word *matka* ‘mother’ look like. (For comparison with the “linguistic” paradigm, see Table 1.1).

Positive attributes of the approach are easy implementation, high speed and a well readable morphology specification.

2.2 Unsupervised algorithms

This section presents two representatives of the unsupervised approaches: Goldsmith’s Linguistica (Goldsmith, 2001), which returns partial paradigms called *signatures* and Morfessor (Creutz and Lagus, 2002, 2005, 2007), which induces morphemic segmentation of plain-text corpora. Paramor (Monson, 2009), the unsupervised algorithm I have extended in this thesis, is described separately in Chapter 3.

Case	**Singular**	**Plural**
nom	mat + ka	mat + ky
gen	mat + ky	mat + ek
dat	mat + ce	mat + kám
acc	mat + ku	mat + ky
voc	mat + ko	mat + ky
loc	mat + ce	mat + kách
inst	mat + kou	mat + kami

Table 2.1: Engineering approach paradigm for *matka* ‘mother’.

2.2.1 Linguistica

Goldsmith (2001) presents Linguistica, an unsupervised approach based on minimum description length (MDL). Basic idea of the approach is to find a morphology model which is compact and allows compact representation of the corpus. More precisely, a model that minimises the sum of the compressed size of the corpus and the compressed size of the model. Assumption is that the model will be close to reality due to the economy of languages.

Morphology model consists of a stem list, suffix lists and a set of *signatures*. A signature contains stems and suffixes (more precisely, pointers to the respective lists). Their presence together means that all combinations of stems and suffixes in the signature are possible in the given language. A stem can be present only in one signature, occurrence of a suffix in more signatures is possible. A sample of several signatures (without stems) can be found in Table 2.2.

To be able to analyse a word as a stem and more than one suffix, Goldsmith introduces *complex stems*, which consist of three pointers: to a signature, a stem and a suffix. For example, the word *workings* would be analysed as a complex stem *working* (pointers to: stem *work*, suffix *ing*, signature for *working*) + suffix *s*. The algorithm works roughly as follows:

1. Words are divided into stems and suffixes using probability based heuristics. Signatures are created according to these splits. One of the heuristics is the following: Consider all the tails up to the length 6 and score them by the measure:

   $$\frac{C(n_1 n_2 \dots n_k)}{N_k} \log \frac{C(n_1 n_2 \dots n_k)}{C(n_1)C(n_2)\dots C(n_k)}$$

   where $n_1, n_2 \dots n_k$ are individual characters of the tail, N_k denotes the number of all tails of length k and $C(x)$ denotes count of x. Top 100 tails ranked by this measure are selected as the candidate suffixes.

2. Signatures containing only one stem or only one suffix are discarded.

3. Two types of modifications to the initial morphology are tested and applied if they decrease the description length:
    - Splitting a suffix into two if the suffix is a concatenation of two already known suffixes. For example, the suffix *ings* is split into *ing* and *s*.
    - If all the suffixes of the signature begin with the same letter, the letter can be moved from the suffixes to the stems. For example, *te.ting.tes* may become *e.ing.es*.

An example of Linguistica's output is shown in Table 2.2. Limitation of Linguistica is that it does not capture stem-internal changes and point-of-affixation phonological/graphemic changes. It also disregards the context of the words, which may be a useful source of information, as shown by the work of Yarowsky and Wicentowski (2000), described in Section 2.3.1.

Rank	**Signature**	**#Stems**	**Rank**	**Signature**	**#Stems**
1	NULL.ed.ing	69	16	e.es.ing	7
2	e.ed.ing	35	17	NULL.ly.ness	7
3	NULL.s	253	18	NULL.ness	20
4	NULL.ed.s	30	19	e.ing	18
5	NULL.ed.ing.s	14	20	NULL.ly.s	6
6	's.NULL.s	23	21	NULL.y	17
7	NULL.ly	105	22	NULL.er	16
8	NULL.ing.s	18	23	e.ed.es.ing	4
9	NULL.ed	89	24	NULL.ed.er.ing	4
10	NULL.ing	77	25	NULL.es	16
11	ed.ing	74	26	NULL.ful	13
12	's.NULL	65	27	NULL.e	13
13	e.ed	44	28	ed.s	13
14	e.es	42	29	e.ed.es	5
15	NULL.er.est.ly	5	30	ed.es.ing	5

Table 2.2: Top English signatures produced by Linguistica

2.2.2 Morfessor

Morfessor is a family of algorithms for unsupervised morpheme segmentation developed by Mathias Creutz and Krista Lagus. Their work progressed from the relatively simple Morfessor-baseline (Creutz and Lagus, 2002) to the latest Morfessor-MAP (Creutz and Lagus, 2007) with a relatively complex probabilistic model. Unlike Linguistica, Morfessor is capable of dividing words into an unlimited number of morphs. This is motivated by agglutinative languages such as Finnish, where words composed of high number of morphemes are frequent. In this section, most of the attention will be focused on Morfessor-MAP, as the other models can be viewed as its simplifications. Short description of the baseline algorithm will be presented as well to demonstrate the basic idea.

Morfessor-baseline uses MDL to evaluate its analyses. It starts with each word as a separate morph. Then for each word, all possible splits into two parts are examined. If one of the parts is an already existing morph and the split decreases the compressed corpus size, the split is accepted and the splitting attempts continue recursively. When the whole corpus is analysed in this way, randomly selected words are merged and resplit.

Such an approach doesn't take into account the morph's function as a prefix, stem or suffix. Therefore, after seeing *write* in the corpus and analysing *writes* as *write* + *s*, *s* is recognised as a morph which enables the analysis of *storm* as *s* + *torm*. To address this issue, later versions of Morfessor use HMM to assign function tags to morphs. There are four such tags: *prefix* (PRE), *stem* (STM), *suffix* (SUF) and *non-morpheme* (NON). The nonmorpheme category helps to tackle a common problem of MDL-based approaches – marking frequent syllables or a word fragments as morphemes. Successful identification of such strings allows to keep them internally for the sake of economical representation of the corpus and at the same time dealing with them when constructing the final analysis. Morfessor-MAP's analysis of a word is a hierarchical structure, such as [[straight[for ward]][ness]]. When creating the final analysis, the structure is expanded to the finest resolution not containing nonmorphemes. So, if *for* is tagged as NON, the [[straight forward][ness]] analysis is presented as a result.

Morfessor-MAP uses a generative probabilistic model and Bayesian *maximum a posteriori* (from here comes the MAP acronym) approach to evaluate quality of the analyses and search for the best one. The approach can be described shortly by the following formula:

$$\arg\max_{\mathbb{M}} P(\mathbb{M}|corpus) = \arg\max_{\mathbb{M}} P(corpus|\mathbb{M}) \cdot P(\mathbb{M})$$

$\mathbb{M}$ stands for a model of the language, which expresses properties of grammar and lexicon. In order to use the formula, a prior probability distribution has to be defined for the model, as well as the posterior $P(corpus|\mathbb{M})$. The posterior is defined by the Hidden Markov Model (HMM):

$$P(corpus|\mathbb{M}) = \prod_{j=1}^{W} \left[P(C_{j1}|C_{j0}) \prod_{k=1}^{n_j} P(\mu_{jk}|C_{jk}) P(C_{j(k+1)}|C_{jk}) \right]$$

where W denotes the token count of the corpus, μ_{jk} the k-th morph in the j-th token and C_{jk} its tag.

The prior $P(\mathbb{M})$ takes into account various features of morphs: their (hierarchical) structure, frequency, length, left and right perplexity. The perplexities express unpredictability of the morph's left/right context. The assumption is that prefixes can be attached to a big number of stems and therefore their right perplexity is high and analogically for suffixes and the left perplexity. This assumption leads to the definition of class probabilities $P(C_{jk}|\mu_{jk})$ for PRE and SUF based on right/left perplexity. Details about the generative model can be found in (Creutz and Lagus, 2007, pp. 9–16).

The search for the most probable hypothesis is heuristic and uses the Morfessor-baseline as the initial hypothesis. It can be summarised as follows:

1. Initialisation of segmentation by Morfessor-baseline
2. Splitting of morphs
3. Joining of morphs using a bottom-up strategy
4. Splitting of morphs
5. Resegmentation of the corpus using Viterbi algorithm and reestimation of probabilities until convergence
6. Repetition of steps 3–5 once
7. Expansion of the morph substructures to the finest resolution that does not contain non-morphemes

See Creutz and Lagus (2005) for detailed description of the search procedure.

Evaluation on the morpheme segmentation task using gold standard showed that Morfessor outperforms Goldsmith's Linguistica on both English and Finnish data. As one may expect, on the Finnish corpus the performance gap was wider.

2.2.3 Semi-supervised learning based on Morfessor

Kohonen et al. (2010) improve Morfessor's performance using correctly segmented words (1000+ for English, 100+ for Finnish) as a seed. The seeded algorithm achieved significant improvement in recall for both languages, accompanied by only a slight drop in precision.

Tepper and Xia (2010) use hand-written transformation rules to capture allomorphy and improve Morfessor's segmentation. Output of the Morfessor-MAP algorithm is used as the initial corpus segmentation. The splits are then re-estimated using an HMM with underlying forms of the morphs as a hidden variable. An underlying form is a single representation of a set of allomorphs. The system was tested on English and Turkish and outperformed the Morfessor-MAP baseline on both languages.

2.3 Combination of multiple sources of information

Works by Yarowsky and Wicentowski (2000) and Schone and Jurafsky (2001) aim at using more sources of information such as string similarity and context similarity to create a combined classifier which achieves significantly higher quality than its individual parts. The approach of Yarowsky and Wicentowski (2000) is described in the following section as a representative of this class of approaches.

2.3.1 Yarowsky and Wicentowski (2000)

Yarowsky and Wicentowski (2000) present an algorithm for aligning inflections to lemmas. They evaluate the algorithm on past tense of verbs and show that it is capable of handling irregular forms. The system works iteratively and ranks potential lemmas and morphological tags for each word by 4 different measures:

1. Frequency similarity. This measure computes the word's relative frequencies with respect to its hypothetical inflections given the candidate tag and lemma. The candidate tag scores well if the frequencies fit the expected distributions of tag-to-tag ratios. For example, *singed* as a candidate for past tense of *sing* will receive a low score, as the frequency ratio *singed/sing* is 0.007, while the overall ratio of *VBD/VB* is 0.85. (The ratio *sang/sing* is 1.19.) The estimators of the tag-to-tag ratios can be trained on the regular forms only, as according to the authors' observations, the ratios of the irregular forms follow similar distributions.

2. Context similarity measure assumes that different forms of the same lemma occur in similar context. The context is extracted using "a set of simple regular expressions over small closed-class parts of speech" with an aim to find subjects and objects of the verbs. The context words are lemmatised using the output from the previous iterations, which helps to reduce sparsity.

3. Weighted Levenshtein distance (Levenshtein, 1966), capable of setting different costs for individual characters or character classes. It uses a cost matrix which in the first iteration is set to less penalise changing vowels to other vowels. Alternatively, a 'trained' matrix from a related language can be used as the initial setting. In the following iterations, the matrix is re-estimated from the <inflection, lemma> alignments from the previous iteration.

4. Probabilistic function mapping inflections to lemmas, modelling stem-final phonological changes. The probabilistic model, called MorphTrans, is trained on the output of the previous iteration. In the first iteration, the probabilities depend on the modified Levenshtein distance.

The algorithm requires a number of resources:

1. Large unannotated corpus.
2. List of the language's parts of speech together with their canonical suffixes.
3. List of open class base forms. It is typically obtainable from dictionaries.
4. A rough mechanism for identifying the candidate parts of speech of the remaining vocabulary, not based on morphological analysis.
5. List of consonants and vowels for given language.

Classifier combination techniques are used to merge the results of the four measures. As the iterations proceed, the MorphTrans component gains relative weight as it becomes better trained. The system achieved excellent results on correctly identifying lemmas of English past tense verbs. Its accuracy on regular verbs was almost 100%, while achieving 80% accuracy on the irregular ones.

The approach has shown that classifiers which are working with various sources of information and not accurate individually can be successfully combined into a high-accuracy system. Table 2.3 shows the scores of the candidate lemmas for the word *shook* in the first iteration (except for the last column, which shows the score by the converged MorphTrans model). Although the correct lemma *shake* was ranked first only by one of the four basic measures, it was successfully identified by the overall consensus similarity measure.

Overall Similarity (Iteration 1)				Context Similarity		Frequency Similarity		Levenshtein Similarity		MorphTrans Similarity (1)		MorphTrans Similarity (C)	
shake	.00149	5.5	1	**shake**	.854	share	.073	shoo	.500	shoot	.002593	**shake**	.465578
shoot	.00126	9.3	2	shave	.323	ship	.068	shoot	.333	shoo	.002593	shoot	.001296
ship	.00104	16.3	3	shape	.210	shift	.062	shoe	.310	shock	.000096	shoo	.001296
shatter	.00061	18.9	4	shore	.194	shop	.060	**shake**	.290	short	.000096	shock	.000048
shop	.00094	19.8	5	shower	.184	**shake**	.058	shop	.236	shout	.000095	short	.000048
shut	.00081	20.6	6	shoot	.162	shut	.052	shout	.236	...	...	shove	.000048
shun	.00039	20.7	7	shock	.154	shoot	.051	show	.236	**shake**	.000003	shore	.000048

Table 2.3: Candidate lemmas of the word *shook*. Source: (Yarowsky and Wicentowski, 2000)

However, with highly inflected languages and other parts-of-speech than verbs, using this approach may be problematic. First, the context similarity measure assumes that inflections of the same lemma occur in similar context. Which works well with English verbs, as they have the same subcategorisation requirements in the past and in the present tense. On the other hand, if we consider Czech nouns, their accompanying adjectives exhibit agreement in case and number and therefore will be in a different form for differently inflected nouns (e.g. *zelená tráva* 'green grass' (sg. nom.) vs. *zelenou trávou* (sg. ins.)).

Second, providing a list of all parts-of-speech and their canonical suffixes for languages with rich inflection would be more labour intensive as there can be hundreds or thousands of tags in the given language and less helpful, as there is great suffix homonymy in such languages. For example, Feldman and Hana (2010) illustrate the homonymy of Czech using the suffix *a*, which has “about 19 different meanings.” (See Table 2.4)

Form	**Lemma**	**Gloss**	**Category**
měst-a	město	town	noun neuter sg gen
			noun neuter pl nom (voc)
			noun neuter pl acc
tém-a	téma	theme	noun neuter sg nom (voc)
			noun neuter sg acc
žen-a	žena	woman	noun feminine sg nom
pán-a	pán	man	noun masculine anim sg gen
			noun masculine anim sg acc
ostrov-a	ostrov	island	noun masculine inanim sg gen
předsed-a	předseda	president	noun masculine anim sg nom
vidě-l-a	vidět	see	verb past feminine sg
			verb past neuter pl
vidě-n-a			verb passive feminine sg
			verb passive neuter pl
vid-a			verb transgressive masculine sg
dv-a	dv-a	two	numeral masculine sg nom
			numeral masculine sg acc

Table 2.4: Homonymy of the *a* ending in Czech (from Feldman and Hana (2010))

2.3.2 Extended MorphTrans model as a supervised learner

Wicentowski (2004) extends the probabilistic model developed in (Yarowsky and Wicentowski, 2000) to handle prefixation and stem-internal vowel change. The extended model is called the WordFrame (WF) and it is used for supervised learning with training data in the form of <*inflection, lemma*> pair list. Optional resources are a prefix list and a suffix list. Inflection is modelled as 5 possible transformations of the lemma:

1. Adding a prefix ψ'_p from the prefix list.

2. Point-of-prefixation stem change $\delta'_p \to \delta_p$

3. Stem internal vowel change $\delta'_v \to \delta_v$. δ'_v and δ_v may contain more than one vowel, but they must be both non-empty or both empty.

4. Point-of-suffixation stem change $\delta'_s \to \delta_s$

5. Removing a canonical suffix (e.g. infinitival ending for verbs) ψ_s and adding a suffix ψ'_s from the suffix list.

Table 2.5 shows differences between the MorphTrans and the WordFrame model on the example of *kept* $\rightarrow$ *keep* transformation. The WordFrame is capable of capturing the stem internal change $e \rightarrow ee$. Such rule is a better generalisation than the stem final change $p \rightarrow ep$, for example it would work for *met* $\rightarrow$ *meet*.

	ψ'_p	$\delta'_p \rightarrow \delta_p$	γ_p	$\delta'_v \rightarrow \delta_v$	γ_s	$\delta'_s \rightarrow \delta_s$	$\psi'_s \rightarrow \psi_s$
MorphTrans	N/A	N/A	N/A	N/A	ke	p $\rightarrow$ ep	t $\rightarrow \lambda$
WordFrame			k	e $\rightarrow$ ee	p		t $\rightarrow \lambda$

Table 2.5: Mapping *kept* $\rightarrow$ *keep* analysed by MorpTrans and WordFrame models. Source: Wicentowski (2004)

As a whole, inflection is a transformation $\delta_p\gamma_p\delta_v\gamma_s\delta_s\psi_s \rightarrow \psi'_p\delta'_p\gamma_p\delta'_v\gamma_s\delta'_s\psi'_s$ where $(\gamma_p\delta_v\gamma_s,\ \gamma_p\delta'_v\gamma_s)$ is the *WordFrame*, the longest common substring with at most one vowel change. Division of the training pairs into the subparts is done deterministically by:

1. Removing the longest matching prefix and suffix present in the affix lists from the inflection and a canonical prefix from the stem.

2. Finding the WordFrame.

3. Remaining strings on the left and right from the WF are considered δ_p/δ'_p and δ_s/δ'_s.

Analysis of an unseen inflection is done probabilistically by finding the lemma maximising

$$P(\delta_p\gamma_p\delta_v\gamma_s\delta_s|\delta'_p\gamma_p\delta'_v\gamma_s\delta'_s) = P(\delta'_v \rightarrow \delta_v, \delta'_p \rightarrow \delta_p, \delta'_s \rightarrow \delta_s|\delta'_p\gamma_p\delta'_v\gamma_s\delta'_s)$$

Point-of-suffixation change probability $P(\delta'_s \rightarrow \delta_s|\delta'_p\gamma_p\delta'_v\gamma_s\delta'_s)$ is taken from a hierarchically smoothed suffix trie created in the training phase. $P(\delta'_p \rightarrow \delta_p|\delta'_p\gamma_p\delta'_v\gamma_s)$ is retrieved from an analogous prefix trie and the vowel change probability is estimated without regard on the local context (by $P(\delta_v|\delta'_v)$).

The approach was tested on inflected verb forms in 32 languages, including Czech, with very good results. The size of the training data ranged from 9 lemmas and 201 inflections for Greek to 5715 lemmas and 23786 inflections for Czech.

2.4 Automatic lexicon acquisition for provided paradigms

The resource-light system by Feldman and Hana (2010) uses manually specified paradigms as an input. The paradigms are specified by suffixes and point-of-affixation stem change rules. For each word in the corpus, possible paradigms are found and only the ones supported by the highest number of forms-tokens and/or form-types are retained.

2.5 Semi-supervised systems using human elicitation

Oflazer et al. (2001) describe a framework for creating morphological analysers of under-resourced languages. The framework makes use of direct human supervision in combination with machine learning in an iterative process.

Input required from a human is a definition of paradigms. A paradigm is a set of words with (almost) the same inflectional behaviour. Each paradigm contains a primary example, optional secondary examples and a lexicon. An example is defined by its *citation form* (the form one would look for in a dictionary) and inflected forms together with their morphological categories. The primary example must provide inflected forms for all values of relevant categories. This is not required for the secondary examples, which may be used to show irregularities for specific category combinations. A paradigm's lexicon contains citation forms of words which (presumably) belong to the paradigm.

After paradigm definition, the examples are automatically segmented into stem and affixes. (Settings described in the paper allow at most one prefix and one suffix for a stem.) All string-prefixes (substrings beginning with the first letter) of a citation form (CF) are considered candidates for the stem. The candidate minimising the sum of edit distances to all the inflected forms is chosen as a stem *s*. Then for each inflected form (IF), the projection of the stem is found (a substring with minimal edit distance to *s*). The remaining strings at the start and the end of the IF are considered prefix/suffix. The resulting segmentation is *(prefix) + CF + (suffix)*. (The citation form is used instead of the stem.)

Segmented lexical forms are then aligned to the surface form as in the example (p. 71):

```
un + happy + est   shop0 + ed
un 0 happi 0 est   shopp 0 ed
```

Alignment has 3 constraints :

> (i) a + in the segmented lexical form is always aligned with an empty string on the surface side, notated by 0; (ii) a consonant on one side is always

aligned with a consonant or 0 on the other side, and likewise for vowels;
(iii) the alignment must correspond to the minimum edit distance between
the original lexical and surface forms. (pp. 70 – 71)

Aligned pairs are used by a transformation based learner to learn context-based rewrite rules which transform the lexical forms into the surface forms. The rules are in form `u -> v || lc _ rc`, where `u` is the symbol in the lexical form, `v` is the symbol in the surface form, `lc` and `rc` regular expressions describing left/right context of a limited length. For the previous examples, one can get rules like:

```
y -> i || p _
y -> i || p _ + e
y -> i || p _ + e s
+ -> 0 || # u n _ h a p
```

(p. 71), where # denotes a word boundary, which may be used in context definition as well. The system allows assigning categories to characters and creation of more general rules by using categories in context definitions. In the presented implementation, characters are marked as consonants or vowels and for each rule, its generalised versions is generated. For example, the rule `0 -> p || p _ + e` (p. 72) can be generalised by 3 different rules:

```
0 -> p || CONSONANT _ + e
0 -> p || p + _ VOWEL
0 -> p || CONSONANT _ + VOWEL
```

The same rule is often generated from more than one example. Number of such examples for a rule is called rule's *promise*.

Rule selection is done iteratively by selecting the best rule and applying it to the list of lexical forms. Rules are sorted according to their promise, with the exception of the segmentation rules (`+ -> 0`), which are put to the bottom of the list. (The morpheme boundary is an important context for most of the rules and should not be removed prematurely.) The topmost rule, which does not cause errors (its application does not make any lexical form to be more distant from its corresponding surface form) is selected and applied to the lexical forms. The process is iterated until all the lexical forms are transformed to their corresponding surface forms.

Using XRCE finite-state tools, the selected rules are compiled together with the lexicon to a finite-state transducer for the paradigm. The tools can be used for testing the analyser and finding where it makes mistakes. It is possible to generate all inflected forms for a citation form or all forms with one value of a category.

When the analysers for paradigms are unioned to create the final analyser, a more elaborate method of testing is used. An error-tolerant finite-state recognizer engine is employed to find candidates for fixing – words in a corpus which are not accepted by the analyser, but close to an accepted form. Then the human may adjust the system by adding more examples in an attempt to fix the errors and next iteration of learning is started.

Chapter 3

Paramor

In this chapter, I will describe Paramor, the system I have selected to build upon in this thesis. Paramor was introduced by Christian Monson in his PhD thesis (Monson, 2009). Its goal is unsupervised discovery of inflectional paradigms and using them for morpheme segmentation.

To model partial paradigms, Monson uses *schemes*. A scheme is defined by a set of candidate suffixes (c-suffixes, see Section 3.1.1). The set of candidate stems (c-stems) adherent to a scheme is obtained deterministically by selecting all c-stems which can form a word (present in the corpus) with each of the scheme's c-suffixes. This is the main difference between schemes and Goldsmith's signatures, where each stem can be assigned only to a single signature.

Definition of schemes implies that with adding more c-suffixes into a scheme, number of c-stems can only drop or stay the same. Schemes form a lattice when we consider partial ordering defined by c-suffix set inclusion. Figure 3.1, taken from (Monson, 2009), illustrates a part of a scheme lattice for an English corpus. The highlighted scheme, (*0, ed, ing, s*) has 106 adherent c-stems. Adding the c-suffix *ly* causes a drop to only 4 c-stems, while removing the c-suffix *s* increases the c-stem count to 201.

3.1 Steps in Paramor's pipeline

3.1.1 Initialisation

In the first step, sets of candidate stems (c-stems) and candidate suffixes (c-suffixes) are populated by considering every possible split into stem and suffix for every word in the corpus.

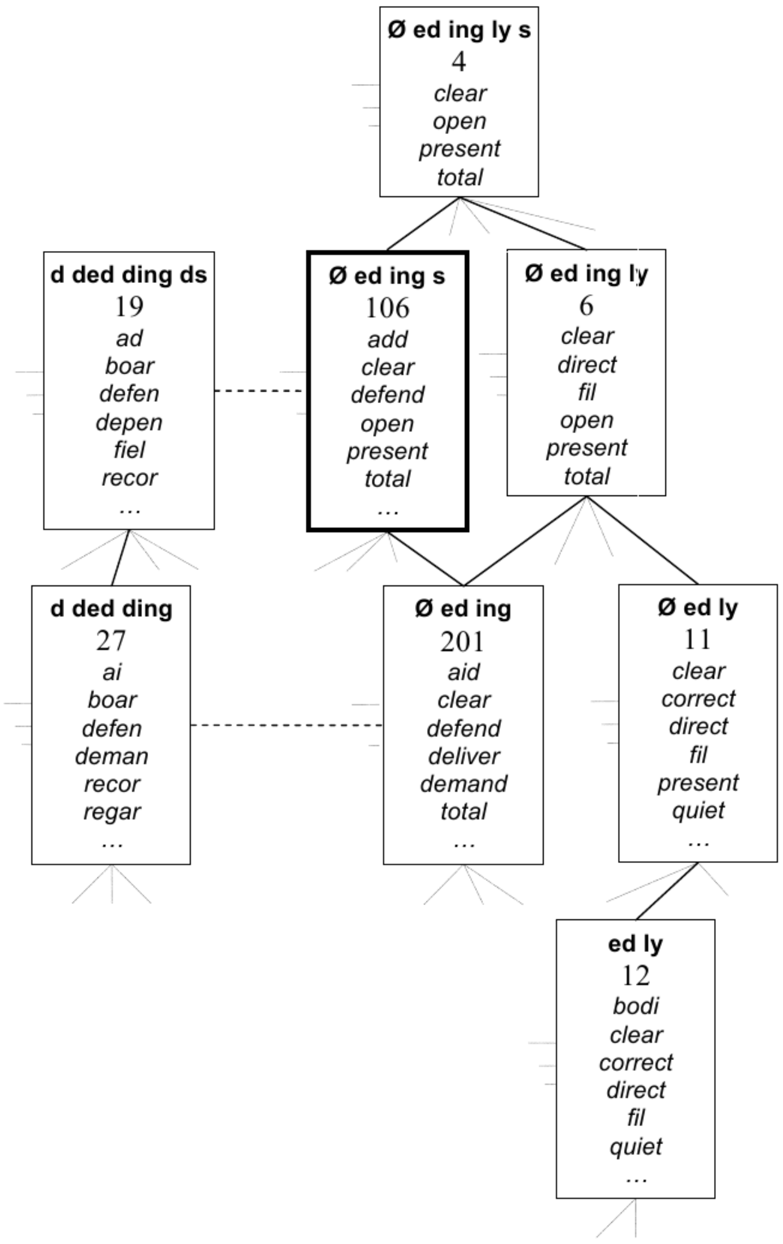

Figure 3.1: Example part of a scheme lattice. Source: (Monson, 2009)

3.1.2 Bottom-up Search

Next phase of the algorithm performs a bottom-up search in the scheme lattice, starting with schemes containing exactly one c-suffix. For each of them, Paramor ascends the lattice, adding one c-suffix at a time until a stopping criterion is met. C-suffix selected for adding is the one with the biggest c-stem ratio. C-stem ratio is the ratio between the number of c-stems in the candidate higher-level scheme and the current scheme. When the highest possible c-stem ratio falls under 0.25, search stops.

For illustration, let's use the lattice from Figure 3.1 and assume that the search has reached the (*0, ed, ing*) scheme, which has 201 adherent c-stems. There are two schemes the search can continue to: (*0, ed, ing, ly*) with 6 c-stems and (*0, ed, ing, s*) with 106 c-stems. The algorithm selects the scheme with the highest number of c-stems – (*0, ed, ing, s*) and checks whether the c-stem ratio is above 0.25. The ratio is 106 / 201 = 0.53 and the search moves to the highlighted scheme. Then there is only one possibility to continue: (*0, ed, ing, s, ly*) with only 4 c-stems, which is rejected due to the c-stem ratio 4 / 106 = 0.04.

3.1.3 Scheme Clustering

Resulting schemes are then subjected to agglomerative bottom-up clustering. To determine proximity of two clusters, sets of (word) types generated by the clusters are measured by cosine similarity.[1] Cluster generates a set of types which is union of sets generated by the schemes it contains. In order to be merged, clusters must satisfy some conditions, e.g. for any two suffixes in the cluster, there must be a stem in the cluster which can combine with both of them.

3.1.4 Pruning

After clustering phase, there are still too many clusters remaining and pruning is necessary. In the first pruning step, clusters which generate only small number of types are discarded. Then Paramor tries to identify clusters modelling incorrect morpheme boundary in a Harrisian fashion by using letter entropy.

3.1.5 Segmentation

Remaining clusters can be used to segment the 'training' corpus or a previously unseen text. For every word in the text, every possible division between stem t and suffix f is examined. If there is a cluster containing f and another suffix f' such that $t.f'$ is a word from the training corpus or the text, Paramor declares morpheme boundary between t

[1] $\text{proximity}(X,Y) = \frac{|X \cap Y|}{\sqrt{|X||Y|}}$

and f. In this way, more than one morpheme boundary may be found in single word. For example, the algorithm analyses *talks* as *talk* + *s* using the scheme cluster (0, *s*, *ing*) if *talk* or *talking* is found in the corpus.

3.2 Results and discussion

Paramor took part in Morpho Challenge 2008[2] with excellent results, especially when combined with Morfessor. Both algorithms achieved precision higher than recall, so a natural idea was to design a combined analyser, returning the union of Paramor's and Morfessor's hypothesised morpheme boundaries. The combined analyser outperformed the other competitors in all 5 of the test languages.[3]

Useful feature of Paramor is that besides being able to segment texts, it outputs the scheme clusters in a human-readable way. Examining the clusters may help getting basic insight into grammatical structure of an unknown language.

A certain disadvantage of Paramor is that it does not handle the phonological/graphemic changes triggered by the affixation, such as the ones shown in Table 1.1. Thus, it faces a similar problem as the engineering approach to morphology (see Section 2.1.2) – the need to create much larger number of paradigms. Unlike in the engineering approach, where paradigms are entered manually, Paramor must find the evidence for the paradigms in the corpus. The richer the inflection of the language is, the more problematic sparsity becomes. An attempt to improve the algorithm by working with allomorphic variants of stems is presented in the following chapter.

[2] http://research.ics.tkk.fi/events/morphochallenge2008/

[3] In Competition 1, which directly evaluated placing of the morpheme boundaries. In Competition 2, which applied the analyses in an IR system, Paramor+Morfessor won in 1 of the 3 languages.

Chapter 4

Modifications of Paramor

Modifications of Paramor were conducted with two motives:

1. Create a semi-supervised system, using manually provided seed of inflected words divided into stems and suffixes.

2. Take into account basic allomorphy of stems.

The modifications were realised by directly changing Paramor's code, rather than by pre-/post-processing. Figure 4.1 shows phases of Paramor on the left with dashed boxes representing my alterations.

In the bottom-up search phase and the scheme cluster filtering phase, the manually provided examples of valid suffixes and their grouping to sub-paradigms are used to steer Paramor towards creating more adequate schemes and scheme clusters. If the seed contains allomorphic stems, they are used to induce simple stem rewrite rules. Using these rules, part of the allomorphic stems in the corpus can be discovered, which helps to find more complete schemes.

4.1 Scheme seeding

The manual seed contains a simple list of inflected words with marked morpheme boundary. English inflections

talk, talks, talked, talking
stop, stops, stopped, stopping
chat, chats, chatted, chatting

would be captured as:

talk, stop/stopp, chat/chatt + 0, s / ed, ing

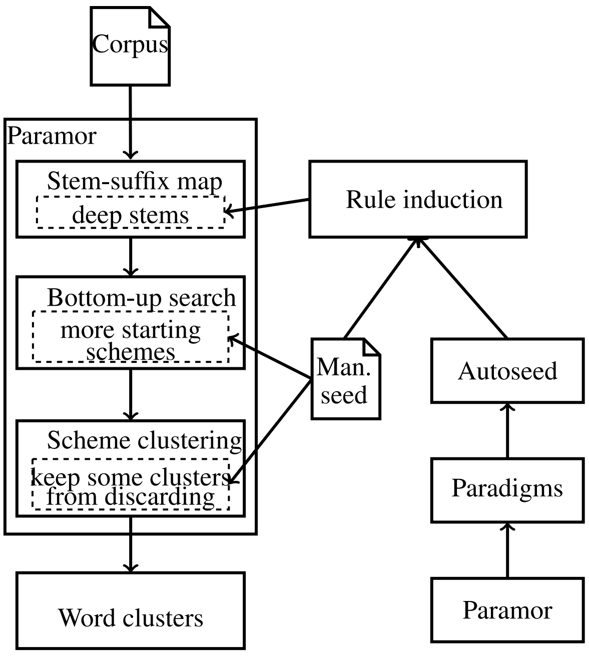

Figure 4.1: Altered Paramor's pipeline

The format of the seed and the seeds used in the evaluation are described in the appendix B. The data are used to enhance Paramor's accuracy in discovering the correct schemes and scheme clusters in the following way:

1. In the bottom-up search, Paramor starts with single-affix schemes. I added a 2-affix scheme to the initial scheme set for every suffix pair from the manual data belonging to the same inflection. Note that one cannot simply add a scheme containing all the suffixes of the whole paradigm as many of the forms will not be present in the corpus.

   Thus, for *talk + 0, s, ed, ing*, one would add the following schemes to the initial scheme set: *(0, s), (0, ed), (0, ing), (s, ed), (s, ing), (ed, ing)*

2. Scheme clusters containing suffixes similar to some of the manually entered suffix sets are protected from the second phase of the cluster pruning. More precisely, a cluster is protected if at least half of its schemes share at least two suffixes with a particular manual suffix set.

   For example, a scheme cluster with suffixes (*í, ích, ímu, ími, ě, ím, ího, ictví*) induced from a Czech corpus was protected from discarding because it contained the suffixes (*í, ím, ího, ímu*), present in one of the examples in the Czech seed – (*letn + í, ím, ího, ímu*) 'summer' masc. sg. adjective, nom, ins, gen/acc, dat respectively.

4.2 Allomorphy

Paramor ignores allomorphy completely (and so do Linguistica and Mofessor). There are at least two reasons to handle allomorphy. First, linguistically, it makes more sense to analyse *winning* as *win+ing* than as *winn+ing* or *win+ning*. For many applications, such as information retrieval, it is helpful to know that two morphs are variants of the same morpheme. Second, ignoring allomorphy makes the data appear more complicated and noisier than they actually are. Thus, the process of learning morpheme boundaries or paradigms is harder and less successful.

This latter problem might manifest itself in Paramor's bottom-up search phase: a linguistically correct suffix triggering a stem change might be discarded, because Paramor would not consider stem allomorphs to be variants of the same stem and c-stem ratio may drop significantly. Further more, incorrect c-suffixes may be selected.

For example, suppose there are 5 English verbs in the corpus: *talk, hop, stop, knit, chat*, together with their *-s* (*talks, hops, stops, knits, chats*) and *-ing* (*talking, hopping, stopping, knitting, chatting*) forms. Let's already have a scheme {*0, s*} with 5 stems. Adding *ing* would decrease the number of stems to 1, leaving only *talk* in the scheme. C-stem ratio would be 0.2 and *ing* would not be accepted. Moreover, incorrect c-suffixes as *ping* and *ting* have c-stem ratio 0.4 and may be accepted.

However, for most languages the full specification of rules constraining allomorphy is not available, or at least is not precise enough. Therefore, I automatically induce a limited number of simple rules from the seed examples and/or from the scheme clusters obtained from the previous run of algorithm. Such rules both over- and undergenerate, but nevertheless they do improve the accuracy of the whole system. For languages, where formally specified allomorphic rules are available, they can be used directly along the lines of Tepper and Xia (2010, 2008), see Section 2.2.3. Currently the system considers only stem final changes, namely vowel epenthesis (e.g., *matk+a* – *matek+0*) and alternation of the final consonant (e.g., *matk+a* – *matc+e*).

4.2.1 Stem change rule induction and application

Formally, the process can be described as follows. From every pair of stem allomorphs in the seed, $a\delta_1, a\delta_2$, where a is their longest common initial substring[1] with suffix sets F_1, F_2 a correspondence rule is generated $*\delta_1 \leftrightarrow *\delta_2$ / (F_1, F_2). A rule $*\delta_1 \leftrightarrow *\delta_2$ / (F_1, F_2) is applicable on an unordered pair of c-stems $x\delta_1, x\delta_2$ present in the corpus if:

1. C-suffix set of the c-stem $x\delta_1$ contains at least one of the suffixes from F_1 and contains no suffix from F_2.

[1]should δ_1 or δ_2 be 0, one final character is removed from a and prepended to δ_1 and δ_2

2. C-suffix set of the c-stem $x\delta_2$ contains at least one of the suffixes from F_2 and contains no suffix from F_1.

For example, from a seed entry

politik/politic + *a, u, ovi, em, y, ů, ům* / *i, ích*

the following rule is generated:

$*k \leftrightarrow *c$ / ($\{$*a, u, ovi, em, y, ů, ům*$\}$, $\{$*i, ích*$\}$)

The rules are used to generate underlying form of c-stems, which I call *deep* stems. I define relation $\leftrightarrow$ between two c-stems: $s_1 \leftrightarrow s_2$ iff there is a correspondence between s_1 and s_2 licensed by any rule. The deep stems then correspond to equivalence classes induced by the reflexive and transitive closure of $\leftrightarrow$. Bottom-up search and all the following phases of Paramor algorithm are then using the deep stems instead of the surface ones.

4.2.2 Stem change rule induction from scheme clusters

In addition to deriving allomorphic rules from the manual seed, I also tested a heuristic for detecting stem allomorphy in the scheme clusters obtained from the previous run of the algorithm. Stem allomorphy increases the sparsity problem and might prevent Paramor from finding complete paradigms. However, if the stem changes are systematic and frequent, Paramor does create the appropriate scheme clusters, although it considers the changing part of the stem to be a part of suffixes.

As an example, consider again the declension of the Czech word *matka* 'mother' in Table 1.1. Paramor's scheme cluster with suffixes $\{$*ce, ek, ka, kami, kou, ku, ky, kách, kám*$\}$ has correctly discovered 9 of the 10 paradigm's suffixes,[2] but fused together with parts of the stem. (Practically discovering the "engineering" paradigm from Table 2.1) Presence of such scheme cluster in the Paramor's output is a hint that there may be a *c/k* alternation and epenthesis in the language. The heuristic presented in this section tries to find scheme clusters with similar characteristics as the $\{$*ce, ek, ka, ...*$\}$ cluster and create stem variants by moving the morpheme boundary.

In the first phase, each scheme cluster with a c-suffix set F is tested by the following procedure:

1. If F contains a c-suffix without a consonant, return *false*.
2. c_c = count of unique left-most consonants found in the c-suffixes in F.
3. If $c_c > 2$ return *false*. If $c_c = 1$ and F doesn't contain any c-suffix starting with a vowel, return *false*.

[2]Except for vocative case singular, which is rarely used.

4. Return *true*.

If a scheme cluster passes this test, each of its stems' subparadigms is examined. Subparadigm for stem s consists of s and F_s – all the c-suffixes from F with which s forms a word in the corpus. For example, let's have a stem $s = mat$ with $F_s = \{ce, ek, ka, ku, ky\}$. Now, the morpheme boundary is shifted so that it is immediately to the right from the first consonant of the original c-suffixes. In the *matka* example, 3 stem variants emerge after the shift:

> *matk + a, u, y*
> *matc + e*
> *matek + 0*

To reduce the amount of falsely detected phonological changes, each stem variant's suffix set is checked whether it contains at least one of the c-suffixes that Paramor has already discovered in other scheme clusters. If the condition holds, rules with the same syntax as the manual seed are created. For example,

> *matk / matc / matek + a, u, y / e / 0*

All generated rules are gathered in a file and can be used in the same way as the manual seed or just for the induction of phonological rules.

Automatically discovered rules for Czech

The heuristic, when tested on scheme clusters generated by Paramor for a Czech corpus with 25k word types, was able to detect a number of phonological/graphemic changes. Examples of detected palatalisation are shown in Table 4.1. Table 4.2 presents some of the detected cases of epenthesis.
There were also incorrect rules generated, for example *poklesl* + *a* $\rightarrow$ *poklesn* + *e*, which relates fem. past participle and 3p. sg. future forms of *poklesnout* 'to decrease'. There is no alternation $l \rightarrow n$ in Czech and the Paramor's original analysis *pokles* + *la*, *pokles* + *ne* was correct.

4.3 Rules for inflectional prefixes

I have also enabled supplying a list of the inflectional prefixes of the given language to Paramor. This feature was tested on Czech, which has only two inflectional prefixes (negative prefix *ne* and superlative prefix *nej*). The decision which prefixes to consider inflectional and which not is to a certain degree an arbitrary decision (e.g., it can be argued that *ne* is a clitic and not a prefix), therefore it makes sense to provide such information manually.

*kni**h*** + *a* ‘book$_{\text{nom.sg.}}$’	$\rightarrow$	*kni**z*** + *e* ‘book$_{\text{loc.sg.}}$’
*strán**k*** + *a* ‘page$_{\text{nom.sg.}}$’	$\rightarrow$	*strán**c*** + *e* ‘page$_{\text{dat.sg.}}$’
*atmosfé**r*** + *a* ‘atmosphere$_{\text{nom.sg.}}$’	$\rightarrow$	*atmosfé**ř*** + *e* ‘atmosphere$_{\text{loc.sg.}}$’
*uměle**c*** + *ký* ‘artistic$_{\text{nom.sg.}}$’	$\rightarrow$	*uměle**č*** + *tí* ‘artistic$_{\text{nom.pl.}}$’
*vojen**s*** + *ký* ‘military$_{\text{nom.sg.}}$’	$\rightarrow$	*vojen**š*** + *tí* ‘military$_{\text{nom.pl.}}$’

Table 4.1: Examples of detected palatalisation.

volb + *a* ‘choice$_{\text{nom.sg.}}$’	$\rightarrow$	*vol**e**b* + *0* ‘choices$_{\text{gen.pl.}}$’
poplatk + *y* ‘fees$_{\text{nom.pl.}}$’	$\rightarrow$	*poplat**e**k* + *0* ‘fee$_{\text{nom.sg.}}$’
pohádk + *y* ‘fairy tales$_{\text{nom.pl.}}$’	$\rightarrow$	*pohád**e**k* + *0* ‘fairy tales$_{\text{gen.pl.}}$’
požadavk + *ům* ‘requests$_{\text{dat.pl.}}$’	$\rightarrow$	*požadav**e**k* + *0* ‘request$_{\text{nom.sg.}}$’

Table 4.2: Examples of detected epenthesis.

Chapter 5

Clustering Framework

The second software package developed for this thesis is a framework for bottom-up clustering. Although its primary purpose is creating and evaluating clusters of morphologically related words, it is general and allows clustering of any objects (not only strings) and using custom distance metrics.

The bottom-up clustering algorithm is simple: At the beginning, one cluster is created for each object. Then, until the stopping criterion is met, the two clusters with the smallest distance between them are merged in each step. Stopping criteria supported by the framework are reaching a predefined number of clusters or the smallest distance between clusters growing above a given threshold.

Once the distance measure between two objects is defined, there is a number of ways to define distance between clusters of objects. Currently, the framework supports 3 of them:

1. Nearest member. Distance between clusters A and B is defined as the **smallest** distance between a member of A and a member of B.

2. Furthest member. Distance between clusters A and B is defined as the **largest** distance between a member of A and a member of B.

3. Average distance. Distance between clusters A and B is defined as the **average** distance between a member of A and a member of B.

5.1 Application to morphology

To use the framework for creating clusters of morphologically related words, it is necessary to define a distance measure between two words. The framework contains 3 pre-built distance measure types. The first one is based on modified edit distance and the other two allow results of unsupervised learners to be used in the clustering. The measures are described below. Euclidean combination of any number of distances is also supported.

5.1.1 Edit distance

As the basic form of the edit distance, the framework contains the Levenshtein distance (Levenshtein, 1966), a distance recognising three basic operations on a string: deletion, insertion or substitution of a single character. By default, each operation has the same cost contributing to the overall distance.

The distance metric can be customised by providing a cost matrix for substitution of characters. This matrix can, for example, represent the fact that some phonemes (represented by graphemes) share more phonological features than others. In such a case, it would make sense to decrease for example the cost of the *s/z* substitution. To design such matrix, one must have quite a detailed knowledge of given language's phonology and orthography. A more resource-light (but less linguistically adequate) approach would just decrease the distance between vowels and between characters differing only in a diacritic marks, such as *a/á* in Czech or *a/ä* in German.

Besides the customisable Levenshtein distance, the framework also contains its modification in which the cost of an operation linearly decreases with the position in the string where it occurs. This distance can serve as a very simple model of suffix-based morphology. In the experimental evaluation, I have used this distance in combination with the above-mentioned approach lowering the costs for diacritics adding/removing and vowel changes. Table 5.1 shows the concrete costs I have selected for use in the experiments.

Change	**Cost**
vowel $\leftrightarrow$ vowel	0.7
with diacritic $\leftrightarrow$ w/o diac.	0.5
other	1.0

Table 5.1: Edit costs used in the experiments.

5.1.2 Segmentation distance

To be able to employ morpheme segmentation algorithms such as Morfessor, I defined a string distance metric based on the words' morphemic segmentation. For a word with m_1 and a word with m_2 morphemes, the distance is: $1 - \frac{2c}{m_1+m_2}$, where c is the length of the longest common initial sequence of morphemes. That means that if the words don't share the initial morpheme, the distance is 1. This definition of the distance is aimed at languages with suffixal morphology.

5.1.3 Paradigm distance

For utilisation of Paramor or another paradigm producing algorithm in the clustering, the framework contains so called paradigm distance. For a word w_1 belonging to a set of paradigms P_1 and a word w_2 with paradigm set P_2 it is defined as $1 - \frac{|P_1 \cap P_2|}{\sqrt{|P_1||P_2|}}$. The fraction expresses the cosine similarity between the two paradigm sets.

5.2 Combination of distance metrics

Currently, the framework supports the Euclidean combination of distance metrics, i.e. for metrics $d_1, \ldots, d_n$, the value of the combined distance between w_1 and w_2 is $\sqrt{d_1(w_1, w_2)^2 + \ldots + d_n(w_1, w_2)^2}$.

Chapter 6

Experiments and Results

This chapter provides description of the input data, experiment settings and the methodology used to evaluate the approaches developed in this thesis. It also presents the results of the experiments.

6.1 Corpora and manual seeds

The approach was tested on two Slavonic languages – Czech and Slovene, one Germanic language – German and on Catalan, a member of the Romanic language family. I used the following sources:

- For experiments with Czech, I used two samples from the Prague Dependency Treebank 1.[1]
- The Slovene corpus is a subset of the jos100k corpus V2.0[2] which contains sampled paragraphs from FidaPLUS,[3] a balanced corpus of Slovene.
- As a German corpus, a part of the TIGER corpus was used. The TIGER corpus[4] consists of German news text from the Frankfurter Rundschau newspaper.
- The source of the Catalan data was the Clic-TALP[5] corpus.

The corpora sources and sizes are summarised in Table 6.1. I evaluated the experiments only on types at least 4 characters long to avoid most of the closed-class and irregular words.

Table 6.1 also shows sizes of the manual seeds measured by number of lemmas. The content of the seeds is included in Appendix B. As a Slovak speaker with high

[1]http://ufal.mff.cuni.cz/pdt/
[2]http://nl.ijs.si/jos/jos100k-en.html
[3]http://www.fidaplus.net/Info/Info_index_eng.html
[4]http://www.ims.uni-stuttgart.de/projekte/TIGER/
[5]http://clic.ub.edu/en/what-is-clic

ID	Source	T	L	T ≥ 4	L ≥ 4	Seed
cz1	PDT1	11k	6k	10k	5.5k	18
cz2	PDT1	27k	13k	25k	12k	18
si	jos100k	27k	15.5k	25k	14k	9
de	TIGER	22k	17k	21k	16k	12
ca	Clic-TALP	11k	8k	10k	7k	62

Table 6.1: Corpora used in the evaluation. T – types, L – lemmas, T ≥ 4 – types longer than 4 characters, L ≥ 4 – lemmas longer than 4 characters

familiarity with Czech, I was able to create the Czech seed without using any external sources of information. The seeds for Slovene and German were obtained from various internet sources, mainly Wikipedia, and their completion took approximately 30 minutes each. The author of the Catalan seed is Dr. Hana, the supervisor of the thesis, who also used internet sources and needed about the same time.

6.2 Experiment settings

There is a large number of possible experiment configurations depending on which inputs Paramor uses, whether clustering is employed, what distance measures are used in case it is employed and other factors. In this section, I will describe the settings I have selected for experimental evaluation.

In the first type of experiments, the word clusters from Paramor's output are evaluated directly against lexemes (sets of all inflections of one lemma) from a lemmatised corpus. Experiments of the second type employ the clustering framework, most of them using the paradigm distance (see Section 5.1.3) based on the Paramor's output. Experiments have been assigned abbreviated names, identifying the Paramor setting (if applicable) and the distance measure used in the experiment (if applicable). Names for the Paramor settings are the following:

1. *noseed* – no seed was used. The baseline configuration.
2. *seed* – manual seed for the given language was used.
3. *autoseed* – the heuristic from Section 4.2.2 was used for stem allomorphy detection.
4. *autoasman* – results from the autoseed heuristic were used as the manual seed.
5. *bothseed* – manual seed was used and the allomorphy rules induced from the seed were merged with the ones from the autoseed.

Distance measures used in the clustering experiments:

1. *edit* – modified edit distance, preferring word-final changes, vowel to vowel changes and diacritic adding/removal.
2. *pdgm* – paradigm distance. Requires output of Paramor as its input.
3. *pdgm.edit.eucl* – Euclidean combination of the paradigm distance and the edit distance.

For Czech, an additional Paramor setting was tested: *pref*, in which the two inflectional prefixes, *ne* and *nej* were supplied to Paramor (see Section 4.3).

6.3 Evaluation

To evaluate the algorithms developed in this thesis against a lemmatised corpus, I used two approaches:

1. Cluster-matching evaluation: comparing clusters against the true sets of inflections.
2. The pairwise approach: Checking each word pair whether it belongs into the same cluster and whether it shares lemma.

Both approaches evaluate precision and recall, from which the balanced F-score is computed.

6.3.1 Cluster-matching evaluation

The first method computes precision and recall of the word clusters in the following way: To compute precision, start with $p = 0$. For each word cluster, find a lexeme with the largest intersection. Add the intersection size to p. Precision = p / sum of cluster sizes. For computing recall, start with $r = 0$. For each lexeme, find a word cluster with the largest intersection. Add the intersection size to r. Recall = r / sum of the lexeme sizes.

6.3.2 Pairwise evaluation

In the pairwise approach approach, similar to one used in Snover et al. (2002), every pair of words belongs into one of the 4 categories:

1. True positives (TP). The words are present in a common cluster and belong to the same lemma.

2. False positives (FP). The words are present in a common cluster but do not belong to the same lemma.

3. False negatives (FN). The words belong to the same lemma, but are not present in the same cluster.

4. True negatives (TN). The rest.

Precision is then defined as $\frac{\text{TP}}{\text{TP}+\text{FP}}$ and recall as $\frac{\text{TP}}{\text{TP}+\text{FN}}$.

6.4 Results

In this section, I will present the results of the evaluation for each language, considering the pairwise evaluation method as the measure of quality. Generally, using the seed improves the results for every evaluated language. On the other hand, using the autoseed or combination with the edit distance usually brings only negligible gains.

6.4.1 Czech

The experiments with Czech data were conducted on two differently sized corpora. The results for the smaller **cz1** are shown in Tables 6.4 and 6.5. Tables 6.6 and 6.7 contain the results for the bigger **cz2** corpus. Increasing the size of the corpus improved the results quite significantly. The best result were achieved by configurations with the manual seed and the two inflectional prefixes provided. The edit distance achieved low score due to producing some noisy clusters such as the one in Table 6.2, where forms of 3 lemmas are mixed.

Form	Gloss	Lemma
plátce	$\text{payer}_{\text{nom/acc.sg.;acc.pl}}$	plátce
plátcem	$\text{payer}_{\text{ins.sg.}}$	plátce
plátci	$\text{payer}_{\text{dat/loc.sg.;nom/ins.pl;}}$	plátce
plátno	$\text{canvas}_{\text{nom/acc.sg.}}$	plátno
plátně	$\text{canvas}_{\text{loc.sg.}}$	plátno
plátek	$\text{slice}_{\text{nom/acc.sg.}}$	plátek

Table 6.2: Example of an erroneous cluster produced by edit distance

Positive impact of seeding on the discovery of nominal paradigms can be illustrated by Table 6.3. The table shows the top 5 nominal scheme clusters (SCs) produced with and without seeding for the **cz2** corpus. The scheme clusters are ordered by the total number of word types in the corpus which they generate. A couple of observations can be made from the table:

1. The seeded SCs are in general supported by a higher number of types and the relative rank of the nominal SCs is higher in the *seed* configuration than in the *noseed* configuration. Recognising the allomorphic stems allowed to find more evidence for the nominal SCs in the *seed* configuration.

   For example, let's compare the cluster #9 (no seed) and the cluster #3 (seed). Both are close to the Czech feminine nominal paradigm *žena* 'woman' (but also include c-suffixes belonging to derived adjectives (*ovní, ovních, ovního*)). Table 1.1, showing declension of the word *matka* 'mother' can serve as an example of this paradigm. The singular dative suffix *-e* is missing in the cluster #9. This suffix triggers palatalisation, which causes either stem-final consonant change (*mat**k** +a* $\rightarrow$ *mat**c** + e*) or changing the suffix to *-ě* (*žen +a* $\rightarrow$ *žen + ě*). The second variant of the suffix (*-ě*) is included in both compared clusters. Thanks to recognition of stem allomorphs, the seeded algorithm was able to find the *-e* suffix as well. Thus, the cluster #3 is not only more complete, but is supported by a larger number of word types as the cluster #9.

2. The top 5 SCs created without seeding contain 3 "engineering" paradigms, concretely #5, #19 and #21, while the top 5 SCs created with seeding contain only one such paradigm (#10). Seeding thus helps to place morpheme boundaries more adequately.

   In the engineering paradimgs, c-suffixes contain the changing part of the stems. For example, the cluster #19 would analyse the words *služba* 'service$_{\text{nom.sg.}}$', *služeb* 'services$_{\text{gen.pl.}}$' as *služ + ba*, *služ + eb*, whereas the corect segmentation of the words is *službb + a*, *služeb + 0*.

Main problem of the Paramor-based configurations was relating forms of different parts-of-speech, created by derivation. For example, in the cluster

(*riziko, rizikový, rizikovost, . . .*)

there is a noun, an adjective derived from the noun, and a noun derived back from the adjective. The English translation would be *risk, risky, riskiness*. This type of error can be considered less serious because the forms share the same stem and treating them as inflections would not be as harmful to an IR system as relating words which are completely unrelated in reality.

6.4.2 Slovene

Tables 6.8 and 6.9 show results for the Slovene language. The best performance was achieved by using the manual seed together with stem allomorphy rules from the autoseed, although the autoseed brought only slight improvements. Very good results

Seed	Rank	#Types	Suffixes
No	5	158	ek kem kovou ková kové kového kovém kový kových kovým kovými ku ky ků kům
	7	153	0 a em ovi ové ových ově u y ů
	9	149	0 a ami ou u y ách ám ě ovní ovních ovního
	19	112	ba bami bou bu by bách bám bě eb ební ebních ebního ebním
	21	102	r ra rech rem rové ru ry rů rům ře
Yes	2	557	0 e ech emi i í ích
	3	306	0 a ami e ou u y ách ám ě ovní ovních ovního ovním
	6	241	0 e ech em ové ových ovým ově u y ů ům
	8	167	0 a em i ova ovi u y ů ům
	10	158	ek kem kovou ková kové kového kovém kový kových kovým kovými ku ky ků kům

Table 6.3: Top 5 nominal scheme clusters produced by Paramor for the **cz2** corpus with and without seeding.

were also achieved by Paramor without seeding combined with the edit distance. Similarly as in the Czech experiments, the most common error Paramor made was relating derived forms sharing a stem. For example, the cluster

(*previden, previdna, previdni, previdno*)

connects forms of the adjective *previden* 'careful' and the adverb *previdno* 'carefully'.

6.4.3 German

The results for German are shown in Tables 6.10 and 6.11. German is the only language where the most successful approach was using the edit distance, either alone or combined with the paradigm distance. There are probably two main reasons for that: First, Paramor is unable to handle stem-internal changes such as the German umlaut (*Mutter/Mütter* 'mother/mothers'). Second, Paramor has also trouble with German compounds, which cause creation of schemes as (*0, organisation*) or (*0, gruppe*).

6.4.4 Catalan

The evaluation results for the Catalan language are presented in Tables 6.12 and 6.13. Providing the manual seed helped Paramor with rich inflection of the Catalan verbs.

In the Catalan corups, collocations are joined together by underscores to create single tokens, which leads to lexemes like

(*porta_a_terme, portant_a_terme, portar_a_terme, portat_a_terme, portava_a_terme*)

Experiment	**P-C**	**R-C**	**F1-C**	**P-P**	**R-P**	**F1-P**
noseed	97.78	80.55	88.33	90.61	47.51	62.33
seed	97.28	84.22	90.28	90.14	53.67	67.28
seed.pref	96.98	86.62	**91.51**	88.18	61.23	**72.28**
autoseed	97.62	80.73	88.38	90.61	47.61	62.42
autoasman	96.25	82.23	88.69	87.28	50.22	63.76
bothseed	97.22	84.24	90.27	90.06	53.68	67.27
bothseed.pref	96.93	86.65	91.50	88.09	61.25	72.26

Table 6.4: Direct evaluation of word clusters – results for the **cz1** corpus. P, R, F1 mean precision, recall and F-measure, -C denotes the cluster-matching evaluation, -P denotes the pairwise evaluation.

Experiment	**P-C**	**R-C**	**F1-C**	**P-P**	**R-P**	**F1-P**
edit	89.35	87.10	88.21	68.67	56.93	62.25
pdgm.noseed	96.80	83.93	89.91	84.44	54.59	66.31
pdgm.seed	96.09	87.66	91.68	83.22	61.24	70.55
pdgm.seed.pref	95.25	90.27	**92.69**	77.67	69.94	**73.61**
pdgm.bothseed.pref	95.22	90.29	**92.69**	77.60	69.96	73.58
pdgm.edit.eucl.noseed	97.17	83.60	89.88	86.93	53.11	65.93
pdgm.edit.eucl.seed.pref	96.38	89.09	92.59	86.13	64.09	73.50

Table 6.5: Results of the clustering experiments for the **cz1** corpus

Experiment	**P-C**	**R-C**	**F1-C**	**P-P**	**R-P**	**F1-P**
noseed	96.97	84.23	90.15	87.65	57.75	69.63
seed	96.89	86.97	91.66	87.14	62.43	72.74
seed.pref	96.72	89.65	**93.05**	86.31	71.59	78.26
autoseed	96.82	84.25	90.10	87.18	58.50	70.02
autoasman	95.81	86.41	90.87	82.25	62.21	70.84
bothseed	96.62	87.10	91.62	86.61	63.07	72.99
bothseed.pref	96.27	90.00	93.03	85.65	72.35	**78.44**

Table 6.6: Direct evaluation of word clusters – results for the **cz2** corpus

with lemma *portar_a_terme* ‘carry out’. This decreased the recall of Paramor, which was unable to relate such inflections and also its precision was lowered by creating schemes as (*0, _de_la_generalitat*) and (*0, _de_tarragona*).

Experiment	P-C	R-C	F1-C	P-P	R-P	F1-P
edit	88.06	86.01	87.02	68.25	56.60	61.89
pdgm.noseed	93.51	88.72	91.06	76.44	66.99	71.40
pdgm.seed	92.77	90.44	91.59	75.01	69.72	72.27
pdgm.seed.pref	92.23	93.69	92.95	74.80	81.16	**77.85**
pdgm.bothseed.pref	91.80	93.89	92.84	73.31	81.88	77.36
pdgm.edit.eucl.noseed	94.64	88.07	91.24	82.46	64.30	72.26
pdgm.edit.eucl.seed.pref	94.60	91.40	**92.98**	84.45	71.17	77.25

Table 6.7: Results of the clustering experiments for the **cz2** corpus

Experiment	P-C	R-C	F1-C	P-P	R-P	F1-P
noseed	94.02	94.09	94.05	69.98	80.40	74.83
seed	94.13	95.46	94.79	69.60	82.74	75.61
autoseed	94.00	94.70	94.35	69.82	81.48	75.20
autoasman	93.21	95.26	94.22	61.50	82.74	70.56
bothseed	93.99	95.63	**94.80**	69.65	83.14	**75.80**

Table 6.8: Direct evaluation of word clusters – results for the **si** corpus

Experiment	P-C	R-C	F1-C	P-P	R-P	F1-P
edit	91.33	90.26	90.80	75.01	64.36	69.28
pdgm.noseed	93.40	91.59	92.49	82.36	68.78	74.96
pdgm.seed	93.35	93.56	93.45	82.79	75.10	78.76
pdgm.bothseed	93.43	93.69	**93.56**	83.18	75.36	**79.08**
pdgm.edit.eucl.noseed	94.82	92.17	93.48	88.09	69.92	77.96
pdgm.edit.eucl.seed	94.98	92.01	93.47	88.43	68.28	77.06

Table 6.9: Results of the clustering experiments for the **si** corpus

Experiment	P-C	R-C	F1-C	P-P	R-P	F1-P
noseed	89.81	93.70	91.72	56.02	64.87	60.12
seed	90.24	94.24	**92.19**	54.41	67.19	**60.13**

Table 6.10: Direct evaluation of word clusters – results for the **de** corpus

Experiment	**P-C**	**R-C**	**F1-C**	**P-P**	**R-P**	**F1-P**
edit	92.03	93.99	93.00	65.68	64.09	**64.87**
pdgm.noseed	92.48	92.43	92.46	65.25	57.61	61.19
pdgm.seed	92.04	93.12	92.58	64.19	60.38	62.23
pdgm.edit.eucl.noseed	91.81	93.86	92.83	62.40	64.19	63.28
pdgm.edit.eucl.seed	92.26	93.94	**93.09**	65.36	64.20	64.77

Table 6.11: Results of the clustering experiments for the **de** corpus

Experiment	**P-C**	**R-C**	**F1-C**	**P-P**	**R-P**	**F1-P**
noseed	86.34	93.73	89.89	57.71	68.72	62.74
seed	87.29	94.59	**90.80**	60.84	71.99	65.95
autoseed	86.34	93.73	89.88	57.75	68.72	62.76
autoasman	86.16	93.75	89.80	57.17	68.77	62.44
bothseed	87.28	94.59	90.79	60.88	71.99	**65.97**

Table 6.12: Direct evaluation of word clusters – results for the **cat** corpus

Experiment	**P-C**	**R-C**	**F1-C**	**P-P**	**R-P**	**F1-P**
edit	92.22	89.66	90.92	66.44	49.15	56.50
pdgm.noseed	89.27	94.20	91.67	58.62	70.18	63.88
pdgm.seed	89.89	94.52	92.15	62.15	71.07	66.31
pdgm.bothseed	89.88	94.52	92.14	62.03	71.07	66.24
pdgm.edit.eucl.noseed	89.32	94.26	91.72	58.55	70.35	63.91
pdgm.edit.eucl.seed	89.88	94.66	**92.21**	61.98	71.75	**66.51**

Table 6.13: Results of the clustering experiments for the **cat** corpus

Chapter 7

Conclusion

The thesis presented an extension of a system for unsupervised morphology acquisition and a word clustering framework able to combine the system's output with modified edit distance. Modifications developed in the thesis enable the system to:

1. accept manually provided seed data in the form of inflections with marked morpheme boundary.
2. handle stem allomorphy using rules induced from the seed.

Testing on 4 languages from 3 language families showed that providing a small number of inflections leads to improvement in the performance of the system. The testing also uncovered some of the shortcomings of the approach. There is a number of issues future work can address:

- Currently, the system does not capture stem-internal changes such as German umlaut (*Mutter/Mütter*). Rules for such changes could be induced from the seed, with possibility to limit the characters from which and to which the changes may take place.
- Frequencies of the words in the corpus are ignored, as well as the context they occur in. Intelligent incorporation of statistical, contextual and semantic features of words or morphemes may lead to significantly higher quality of the analysis.

Appendix A

Software package

This appendix describes the usage of the software developed for the thesis. The software has two major parts: modified Paramor and the clustering framework (CF).

A.1 Requirements

For compilation of Paramor and the CF, Java Development Kit (JDK) 6 or higher and the Apache Ant is required. Some scripts in the package require Python 2 in the version 2.5 or higher.

A.2 Installation and usage

To install the SW package, first download the file `klic-morph-sw.zip` from `http://purl.org/klic-morph/download`. Unpack the archive to any selected directory. The structure of the package is described in Section A.2.1. Now it is necessary to compile the Java projects. To compile Paramor, just run `ant` in the directory `paramor_rk`. To compile the CF, run the same command in the directory `clustering`.

A.2.1 Structure of the package

The package consists of the following directories:

- `paramor_rk` The modified Paramor's source code and settings files are stored here.
- `clustering` This directory contains the source code and settings files of the CF.
- `resources` In this directory, corpora and other resources are stored. Currently it contains only the Slovene corpus, which is freely distributable.

A.2.2 Using Paramor

Paramor is located in the directory `paramor_rk`. The original way of using Paramor is interactive, with the user entering commands into Paramor's command line. The commands are described in `README.txt` file. To invoke the interpreter, run

```
java -cp ./bin:./trove.jar -Xmx1g monson.christian.morphology.
paraMor.ParaMor -if <settings file>
```

Settings files

To illustrate the syntax of a settings file, I will show the content of one of the settings files I used in the evaluation:

```
corpus ../resources/cz10/dev37kwPlain.txt

manualSeed seed-cz.txt

autoSeed autoSeed-cz10.txt

prefixList prefixes-cz.txt

language GENERIC

throwOutNumbers on

caseSensitive off

typesToRead 50000
```

The file contains the original Paramor's options, as well as the ones I have added for the extended Paramor features:

- **corpus**: the corpus location, relative to the `paramor_rk` directory. Only plain text corpora are supported.
- **language**: This setting only influences tokenization and should be set to GENERIC.
- **manualSeed**: (optional) location of the manual seed. (See Appendix B for syntax of the seed.)
- **autoSeed**: (optional) location of the autoseed, which will be used to induce stem allomorphy rules.
- **prefixList**: (optional) location of the inflectional prefix list.
- **throwOutNumbers**: (optional) Setting to *on* discards all the tokens containing numerical characters.

- **caseSensitive**: (optional) Setting to *off* makes all the words lowecase.
- **typesToRead**: (optional) Upper limit on the number of different word types to read from the corpus.

Batch mode

To enable running Paramor in a batch mode, I have added the script `paramor.py`, which takes a settings file as its only argument:

```
python paramor.py <settings file>
```

Output files

- **clusterToWords.txt**: the file contains on each line a scheme cluster number, a c-stem and all its inflections according to the scheme cluster. For example:

  ```
  14-return_ returning return returns returned
  ```

  This file is used for evaluation of Paramor against the lexemes in a lemmatised corpus.

- **wordToClusters.txt**: for each word type, the file contains the set of scheme clusters according to which it can be segmented and for each such scheme cluster it shows the type's c-stem e.g.,

  ```
  inquiringly 18-inquir_ 20-inquiring_
  ```

  The file is used for computing the paradigm distance (defined in Section 5.1.3).

- segmented corpus: If Paramor is run in the batch mode, the segmented corpus is stored in a file with a long name, encoding some of the settings Paramors was run with. An example of such name: `segmented-R0.25-ClusterSize20-MBTFLF-ENTROPY-0.5-MBTFRF-ENTROPY-0.5-combinedSegmentation-segmentations.txt`

- scheme clusters: Parmor outputs the resulting scheme clusters to the file **junk-clusters.txt**. In this curiously named file, the scheme clusters are presented together with a tree structure, showing the steps of the bottom-up scheme clustering.

A.2.3 Using the clustering framework

The clustering framework (CF) is located in the `clustering` directory. The CF is primarily a Java library. This section describes usage of the CF for running and evaluating experiments; for examples of usage of the API, see Section A.3. The evaluation is implemented by the `EvalClustering` Java class located in the package `cz.klic.eval`. It performs clustering with the settings specified in a file and evaluates the resulting clusters using a lemmatised corpus. If compiled, the class is runnable by the script `runEval.sh`, which is a shortcut for

```
java -Xmx1g -cp ./bin:trove.jar cz.klic.eval.EvalClustering
```

Usage of the script is:

```
./runEval.sh [-v] <experiment settings file>
```

If the `-v` option is used, the output is more verbose, logging every cluster merge and evaluating each cluster in the result set.

Settings files

The experiment settings files use the syntax of the `.properties` configuration files.[1] As an example, I will show one the files I have used in the evaluation:

```
corpusFile = ../resources/cz10/dev37kw.p3m

corpusFormat = CSTS

distanceMeasure = paradigm, edit

combination = Euclid

clusterApproach = AVERAGE_DISTANCE

distThreshold = 0.99

minClusterCount = 5500

paradigmFile = ../resources/cz10/wordToClusters-noseed.txt
```

The following list describes the options used in the file:

- **corpusFile**: the location of a lemmatised corpus, relative to the `clustering` directory.
- **corpusFormat**: Currently supported values are *CSTS*, *CONLL*, *TIGER_EXPORT* and *SPACE_SEP*

[1] http://en.wikipedia.org/wiki/.properties

- **distanceMeasure**: The distance measure to be used in the clustering. Currently supported values are *paradigm* and *edit*. A comma delimited list of values may be specified. In that case, the **combination** option must be set.
- **combination**: How the metrics specified in the **distanceMeasure** option should be combined. Currently supported values are *Euclid* and *sequence*. The value *sequence* means that the clustering will work step-wise, in each step using different distance metric. Stopping criteria for individual steps can be specified in the **distThreshold** and **minClusterCount** options.
- **clusterApproach**: (optional) The supported values are *NEAREST_MEMBER*, *AVERAGE_DISTANCE*, *FURTHEST_MEMBER* and *NO_CLUSTERING*. The first three values correspond to the approaches described in Chapter 5. The value *NO_CLUSTERING* means that no clustering will be run, only evaluation will be performed for the clusters in the file specified by the **paradigmFile** option. The default value of this option is *AVERAGE_DISTANCE*.
- **distThreshold**: (optional) If the distance between the two closest clusters rises above the given threshold, the clustering stops.
- **minClusterCount**: (optional) If the number of clusters reaches the given number, the clustering stops.
- **paradigmFile**: If the paradigm distance is used, the value should be a file with the syntax of the **wordToClusters.txt** file from Paramor's output (see Section A.2.2). If *NO_CLUSTERING* was specified, the value should be a file with the syntax of the **clusterToWords.txt** file from Paramor's output.

Output of the script

The script `runEval.sh` writes the evaluation results to the standard output. If the `-v` option is used, it reports also each merge of clusters during the run of clustering:

```
...
merge dist:  0.4237868872193668
Merging:
[ekonomiku, ekonomiky, ekonomice]
[ekonomika, ekonomikami]
...
```

and evaluates the resulting clusters by aligning them with the lexeme from the corpus which has the largest intersection. The words which should be added to the cluster and the words which should be removed from the cluster to match the lexeme are shown:

```
...
cluster:studia, studie, studiem, studiu, studium
correct:studia, studiem, studiu, studium
wrong:studie
missed:
cluster:podílel, podílet
correct:podílel, podílet
wrong:
missed:podílí
...
```

A.3 Using the clustering framework API

To run clustering, use the class `cz.klic.clustering.HierarchicalClustering<T>`. `T` is the class of the clustered objects. I used `String` as the value of `T` in the morphology experiments. First, instantiate the class, providing the distance metric and clustering approach to be used, for example:

```
HierarchicalClustering<String> hc = new HierarchicalClustering<String>(
    new LevenshteinMetric(),
    HierarchicalClustering.ClusterApproach.AVERAGE_DISTANCE);
```

You can use one of the metrics already implemented in the `cz.klic.stringDistance` package or implement the `DistanceMetric<T>` interface in the same package.

In the next step, run the clustering itself, using the `cluster` method, which accepts either a list of `T` instances or a list of `Cluster<T>` instances. The optional parameters `clustNum` and `distThreshold` with default values 1 and infinity, respectively, set the stopping criteria. The method returns a list of `Cluster<T>` instances.

```
List<String> words = corpus.getVocab();
List<Cluster<String>> clusters = hc.cluster(words, 1000, 0.95);
```

A list of the members of a `Cluster<T>` instance can be retrieved by the `getMembers` method.

Appendix B

Seeds used in the experiments

This appendix contains the manually provided inflections used as a seed in the experiments described in Chapter 6. Notation works in the following way:

- On the left of the '+' sign, a list of comma delimited stems is entered. On the right side, a list of suffixes is entered. Zero morpheme is denoted by 0. Example:

  walk, talk + 0, s, ed

- If a stem has allomorphic variants, they are entered delimited by slashes (variant1/variant2/...). In such a case, the suffix list must be divided by the same number slashes to contain a sub-list for each stem variant. Example:

  stop/stopp + 0, s / ed

- It is possible to combine allomorphic stems with simple stems. Simple stems are considered being able to combine with all the suffixes. Example:

  walk, talk, stop/stopp + 0, s / ed

B.1 Czech

- Masculine animate nouns; with examples of palatalisation (*k* to *c*, *h* to *z*, *r* to *ř*).
  kluk/kluc, vlk/vlc, politik/politic, logik/logic, vrah/vraz + a, u, ovi, em, y, ů, ům / i, ích
  maďar/maďař + a, u, ovi, em, ech, y, ů, ům / i

- Masc. inanimate nouns:
  hrad, článek/článk + 0 / u, em, y, ům

- Feminine nouns; with an example of epenthesis.
  politik/politic, logik/logic, agentur/agentuř + a, ou, y, 0, ami, ách, ám / e

svadb/svadeb + a, ou, y, ě, ami, ách, ám / 0
klec + 0, í, e, i, ích

- Verbs:
  píš, buduj + i, eš, e, eme, ete, í
  piš, buduj + 0, me, te
  psa + l, la, li, ly
- Adjectives:
  letn + í, ím, ího, ímu
  hrub + ý, ého, ých, á, ou, é, ého, ým

B.2 Slovene

- Masculine nouns:
  korak, vrelec/vrelc + 0 / a, u, om, ov, e, ih, i
- Feminine nouns:
  lip + a, e, i, o, am, ah, ami
  perut + 0, i, jo, im, ih
- Neuter nouns:
  mest + o, a, u, o, om, 0, ih, i
- Verbs:
  kupuj + em, eš, e, emo, ete, ejo
  kupova + l, la, li
  žel + im, iš, i, ita, imo, ite, ijo
- Adjectives:
  dežurn + a, e, i, o, ega, emu, em, im, ima, ih

B.3 German

- Nouns:
  bild + 0, e, es, er, ern
  tisch + 0, es, e, en
  diamant + 0, en
  fahrer + 0, s, n
  buchstab + e, en, ens
  radio + 0, s
  mutter/mütter + 0/0, n

- Adjectives:
  neu + er, en, es, e, em
- Verbs:
  lieb, kauf, arbeit/arbeite + e, en / st, t, te, test, ten, tet
  handel/handl + n, e, st, t, n, te, test, ten, tet / e

B.4 Catalan

- Plural (Nouns, Adjectives)
  roure, fort + 0/s
  balanç/balanc, dolç/dolc, cuc/cuqu, sec/sequ, oblic/obliqü, Pasqu/Pasqü, inicu/iniqü, platj/platg, roj/rog, vag/vagu, amarg/amargu, llengu/llengü, ambigu/ambigüe + a/es
  cantó/canto, ple/ple + 0/ns
  gas, gos/goss, braç, reflex, gris, espès + 0/os
  noi, gat, fill, pare, mare, mar, vent, índex, falç + 0, s
- Nouns in -a: -es
  gat, fill, poet + a, es
- Nouns in stressed vowel (à ó ò ú í é è): -ns
  missió/missio + 0 / ns
  nació/nacio + 0 / ns
  capità/capita + 0 / ns
- Masculines with ”unpronounceable” plural in -s: -os
  braç, pis, nas, peix, sufix, despatx, disc, gest, text + 0, os
  passeig/passej, lleig/lletj + 0 / os
- Adjectives
  blanc/blanqu + 0, a, s / es
  feliç/felic + 0, os / es
  diferent + 0, s
- Verbs Conj. 1
  cant, tanc/tanqu, caç/cac, furg/furgu, raj/rag, obliqu/obliqü + ar, ant, at, o, a, ava, aves, ava, àvem, àveu, aven, ares, à, àrem, àreu, aren, aré, aràs, arà, arem, areu, aran, aria, aries, aria, aríem, aríeu, arien, a / es, em, eu, en, í, i, is, i, em, eu, in, és, essis, és, éssim, éssiu, essin, i, em, eu, in

- Verbs Conj. 2
  perd + re, ent, ut, o, s, 0, em, eu, en, ia, ies, ia, íem, íeu, ien, í, eres, é, érem, éreu, eren, ré, ràs, rà, rem, reu, ran, ria, ries, ria, ríem, ríeu, rien, i, is, i, em, eu, in, és, essis, és, éssim, éssiu, essin, 0, i, em, eu, in
  tém/tem + er / ent, ut, o, s, 0, em, eu, en, ia, ies, ia, íem, íeu, ien, í, eres, é, érem, éreu, eren, eré, eràs, erà, erem, ereu, eran, eria, eries, eria, eríem, eríeu, erien, i, is, i, em, eu, in, és, essis, és, éssim, éssiu, essin, i, em, eu, in

- Verbs Conj. 3
  sent + ir, int, it, o, s, 0, im, iu, en, ia, ies, ia, íem, íeu, ien, í, ires, í, írem, íreu, iren, iré, iràs, irà, irem, ireu, iran, iria, iries, iria, iríem, iríeu, irien, i, is, i, im, iu, in, ís, issis, ís, íssim, íssiu, issin, 0, i, im, iu, in
  acolor + ir, int, it, eixo, eixes, eix, im, iu, eixen, ia, ies, ia, íem, íeu, ien, í, ires, í, írem, íreu, iren, iré, iràs, irà, irem, ireu, iran, iria, iries, iria, iríem, iríeu, irien, eixi, eixis, eixi, im, iu, eixin, ís, issis, ís, íssim, íssiu, issin, eix, eixi, im, iu, eixin

Bibliography

Mathias Creutz and Krista Lagus. Unsupervised discovery of morphemes. In *Proceedings of the ACL-02 workshop on Morphological and phonological learning - Volume 6*, MPL '02, pages 21–30, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics. doi: http://dx.doi.org/10.3115/1118647.1118650.

Mathias Creutz and Krista Lagus. 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)*, pages 106–113. Finland: Espoo, 2005.

Mathias Creutz and Krista Lagus. Unsupervised models for morpheme segmentation and morphology learning. *ACM Trans. Speech Lang. Process.*, 4:3:1–3:34, February 2007. ISSN 1550-4875. doi: http://doi.acm.org/10.1145/1187415.1187418. URL http://doi.acm.org/10.1145/1187415.1187418.

Anna Feldman and Jirka Hana. *A resource-light approach to morpho-syntactic tagging*. Rodopi, Amsterdam/New York, NY, 2010. URL http://www.rodopi.nl/ntalpha.asp?BookId=LC+70.

John A. Goldsmith. Unsupervised learning of the morphology of a natural language. *Computational Linguistics*, 27(2):153–198, 2001.

Jan Hajič. *Disambiguation of Rich Inflection: Computational Morphology of Czech*. Karolinum, Charles University Press, Praha, 2004.

Lauri Karttunen, Kimmo Koskenniemi, and Gertjan van Noord. Finite state methods in natural language processing. *Natural Language Engineering*, 9(1):1–3, 2003.

Oskar Kohonen, Sami Virpioja, and Krista Lagus. Semi-supervised learning of concatenative morphology. In *Proceedings of the 11th Meeting of the ACL Special Interest Group on Computational Morphology and Phonology*, SIGMORPHON '10, pages 78–86, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics. ISBN 978-1-932432-76-3. URL http://portal.acm.org/citation.cfm?id=1870478.1870488.

Kimmo Koskenniemi. Two-level model for morphological analysis. In *IJCAI*, pages 683–685, 1983.

Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19(2): 313–330, 1993.

Christian Monson. *ParaMor: From Paradigm Structure to Natural Language Morphology Induction*. PhD thesis, Language Technologies Institute, School of Computer Science, Carnegie Mellon University, 2009.

Kemal Oflazer, Sergei Nirenburg, and Marjorie McShane. Bootstrapping morphological analyzers by combining human elicitation and machine learning. *Computational Linguistics*, 27(1):59–85, 2001.

Patrick Schone and Daniel Jurafsky. Knowledge-free induction of inflectional morphologies. In *Proceedings of the North American chapter of the Association for Computational Linguistics*, pages 183–191, 2001.

Matthew G. Snover, Gaja E. Jarosz, and Michael R. Brent. Unsupervised learning of morphology using a novel directed search algorithm: Taking the first step. In *In Proc. ACL Worksh. Morphol. & Phonol. Learn*, pages 11–20, 2002.

Michael Tepper and Fei Xia. A hybrid approach to the induction of underlying morphology. In *Proceedings of the Third International Joint Conference on Natural Language Processing (IJCNLP-2008), Hyderabad, India, Jan 7-12*, pages 17–24, 2008. URL `http://www.aclweb.org/anthology-new/I/I08/I08-1003.pdf`.

Michael Tepper and Fei Xia. Inducing morphemes using light knowledge. *ACM Trans. Asian Lang. Inf. Process.*, 9:3:1–3:38, March 2010. ISSN 1530-0226. doi: http://doi.acm.org/10.1145/1731035.1731038. URL `http://faculty.washington.edu/fxia/mpapers/TALIP2010.pdf`.

Richard Wicentowski. Multilingual noise-robust supervised morphological analysis using the wordframe model. 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 70–77, Stroudsburg, PA, USA, 2004. Association for Computational Linguistics. URL `http://portal.acm.org/citation.cfm?id=1622153.1622162`.

David Yarowsky and Richard Wicentowski. Minimally supervised morphological analysis by multimodal alignment. In *Proceedings of the 38th Meeting of the Association for Computational Linguistics*, pages 207–216, 2000.