Inferring Morphological Rules from Small Examples using 0/1 Linear Programming

Inferring morphological rules from small examples using 0/1 linear programming

Ann Lilliestöm, Koen Claessen, Nicholas Smallbone
Chalmers University of Technology
Gothenburg, Sweden
{annl,koen,nicsma}@chalmers.se

Abstract

We show how to express the problem of finding an optimal morpheme segmentation from a set of labelled words as a 0/1 linear programming problem, and how to build on this to analyse a language's morphology. The approach works even when there is very little training data available.

1 Introduction

Many recent tools for morphological analysis use statistical approaches such as neural networks (Cotterell et al., 2018). These approaches can profitably use huge amounts of training data, which makes them ideal for high-resource languages. But if there is little training data available, statistical methods can struggle to learn an accurate model. And when they learn the wrong model, it is difficult to diagnose the error, because the model is a black box.

This paper presents a new approach to morphological analysis that produces human-understandable models and works even if only a few training words are given. The user gives a list of inflected words together with each word's morphosyntactic features and standard form (lemma):

standard	inflected	features
woman	women	Pl;Nom
baby	babies'	Pl;Gen
dog	dogs'	Pl;Gen
cat	cat's	Sg;Gen
lorry	lorries	Pl;Nom

Our tool then proposes, for each feature, the affixes and morphological rules which mark that feature. For the example above it suggests the following:

feature	context	morpheme
Gen	Sg	+'s∗
Gen	Pl	+'∗
Pl	Gen	+s∗
Pl	Nom	+a+ → +e+
Pl		+y∗ → +ies∗
Sg		+y∗, +a+, ∅

Here, +'s∗ represents the *suffix* 's, and +a+ represents an *infix* a.[1] The table shows that both *'s* and an apostrophe can mark the genitive case; the second column means that the genitive was marked by *'s* only in singular nouns, and by an apostrophe only in plural nouns. An *s* suffix marks plural, and because of the tiny input data it was only seen in genitive nouns. Plural can be marked by an inner vowel change from *a* to *e* (indicated by the arrow), or by changing a final *y* to *ies*, in which case the singular form is marked by *a* or *y*.

The tool also segments the input words into morphemes consistent with the table of rules (the inferred stem is marked in bold):

standard	inflected
wom|a$_{\text{Sg}}$|**n**|∅$_{\text{Nom}}$	**wom**|e$_{\text{Pl}}$|**n**|∅$_{\text{Nom}}$
bab|y$_{\text{Sg}}$|∅$_{\text{Nom}}$	**bab**|ies$_{\text{Pl}}$|'$_{\text{Gen}}$
dog|∅$_{\text{Sg,Nom}}$	**dog**|s$_{\text{Pl}}$|'$_{\text{Gen}}$
cat|∅$_{\text{Sg,Nom}}$	**cat**|'s$_{\text{Gen}}$|∅$_{\text{Sg}}$
lorr|y$_{\text{Sg}}$|∅$_{\text{Nom}}$	**lorr**|ies$_{\text{Pl}}$|∅$_{\text{Nom}}$

Our key idea is a novel *morphological segmentation* algorithm used to produce the segmentation above (section 3). Once we have a segmentation, we use it to compute the rules on the left (section 4).

In this example, the rules inferred by our tool precisely capture the morphology of the input words, and the segmentation shows which rules are most common. We claim that, together, they can be used to perform high-quality morphological analysis. We validate this claim in section 5 by showing how to build on our tool to do reinflection.

2 Related Work

Morphological segmentation is a well-studied problem in NLP, with applications including machine translation (Green and DeNero, 2012), speech

[1] The tool also supports prefixes such as ∗un+, and circumfixes such as ∗ge+t∗ in the German *gerannt* (run; past participle). We explain the meaning of ∗ and + in section 3.3.

Proceedings of the 22nd Nordic Conference on Computational Linguistics (NoDaLiDa), pages 164–174
Turku, Finland, 30 September – 2 October,

recognition (Rajagopal Narasimhan et al., 2014) and information retrieval (Turunen and Kurimo, 2008); for a more detailed overview we refer the reader to Ruokolainen et al. (2016) or Hammarström and Borin (2011). Unsupervised learning (Harris, 1955; Creutz and Lagus, 2007; Goldsmith, 2001; Johnson, 2008; Poon et al., 2009) relies on unannotated text, and is perhaps the most popular approach, because of the large amount of unannotated text available for many languages, but it can suffer from low accuracy (Hammarström and Borin, 2011). One way to improve accuracy is to exploit semantic information (Sakakini et al., 2017; Vulić et al., 2017; Schone and Jurafsky, 2000; Soricut and Och, 2015). Another is minimally-supervised learning (Monson et al., 2007; Kohonen et al., 2010; Ruokolainen et al., 2013; Grönroos et al., 2014; Sirts and Goldwater, 2013; Ahlberg et al., 2014), which combines a large amount of unannotated text with a small amount of annotated text, and potentially provides high accuracy at a low cost.

Silfverberg and Hulden (2017) observe that in the Universal Dependencies project (Nivre et al., 2017), each word is annotated with its lemma and features, but not segmented. They study the problem of finding a segmentation for these words. Our segmentation algorithm solves the same problem, and can be used in their setting. We improve on their solution by using a constraint solver to achieve high accuracy even with limited data, and using a precise language for expressing affixes and rules, which allows us to use the resulting model to precisely analyse new words.

Luo et al. (2017) use Integer Linear Programming for unsupervised modelling of morphological families. They use ILP as a component of a larger training algorithm, so unlike our work they do not attempt to find a globally optimal solution. ILP has been used in other NLP applications outside of morphology (Berant et al., 2011; Roth and Yih, 2005; Clarke and Lapata, 2008).

3 Morphological segmentation

The input to our tool is a list of words, each annotated with its lemma and morphosyntactic features. From now on, we model the lemma as simply another feature; for example, the word "women" has the features Nom, Pl and woman.

The goal of this section is to divide each word into segments, and assign each feature of the word to one of those segments. A segment may consist of multiple discontinuous pieces, or be empty. In the segmentation on page 1, for example, the word *women* is segmented by assigning Pl the segment *e*, Nom the null segment, and *woman* the discontinuous segment *wom*n*. In general, the segment assigned to the lemma feature represents the stem of the word.

We say a segmentation is valid if for each word:

- Every letter is in exactly one segment.
- Each feature labels exactly one segment.
- Each segment is labelled with exactly one feature.[2]

There are many valid segmentations, some good, some bad. We consider a segmentation good if it is parsimonious: each morpheme should be marked by as few features as possible. Note that different forms of a word may be assigned different stems (e.g. *go* and *went*), but parsimony dictates that we share stems wherever reasonable. Perhaps surprisingly, finding the most parsimonious segmentation is an NP-hard problem, by reduction from set cover.

3.1 Segmentation as a constraint problem

We solve the segmentation problem using *zero-one linear programming* (0/1 LP). A 0/1 LP problem consists of a set of variables (e.g. $x, y, z, \ldots$) and a set of linear inequalities over those variables (e.g. $2x + 3y \geq z$ and $x \leq 2y + 3$). Given such a problem, a 0/1 LP solver finds an assignment of values to the variables that makes all the inequalities hold, and where all variables have the value 0 or 1. A 0/1 LP problem also specifies a linear term whose value should be minimised (e.g. $4x + y - z$), called the *objective function*. A 0/1 LP solver is guaranteed to find the solution that minimises the objective function, if a solution exists. The solver that we use is CPLEX (ILOG, 2009).

In this section we assume that we are given, in addition to the labelled words, a (possibly large) set of allowable segments for each feature, which we call *candidate morphemes*. How candidate morphemes are automatically generated is explained in section 3.2.

Our encoding uses the following variables. If m is a candidate morpheme of feature f, the variable

[2] This is an unrealistic assumption, but also fairly harmless: when a morpheme marks (e.g.) genitive plural, it will be assigned one of those features, but the inferred rules will show that it only occurs together with the other feature.

$S_{f,m}$ should be 1 if m is used to mark f, and 0 otherwise. If f is a feature with a candidate morpheme m that occurs in word w, and P is the set of positions in w occupied by m, then the variable $M_{w,P,f,m}$ should be 1 if this occurrence of m marks feature f, and 0 otherwise.

Example Suppose that we are given the problem of segmenting the Swedish word *hästarna* (the horses). For simplicity, suppose that we know that the stem of *hästarna* is *häst*, and that the possible affixes are *ar* and *na*. (For now, we ignore the fact that these should be suffixes; we deal with this in section 3.3.) This results in the following candidate morphemes:

feature	candidate morphemes
Pl	ar, na
Def	ar, na
häst	häst

The string *ar* appears at positions 5–6 in *hästarna*, and *na* appears at positions 7–8. Therefore, the encoding introduces the following variables: $S_{\text{Pl,ar}}$, $S_{\text{Pl,na}}$, $S_{\text{Def,ar}}$, $S_{\text{Def,na}}$, $S_{\text{häst,häst}}$, $M_{\text{häst,5–6,Pl,ar}}$, $M_{\text{häst,5–6,Def,ar}}$, $M_{\text{häst,7–8,Pl,na}}$, $M_{\text{häst,7–8,Def,na}}$, $M_{\text{häst,1–4,häst,häst}}$.

We then generate the following constraints:

- *If a morpheme is used to mark a feature in a given word, it must be a genuine morpheme of that feature.* For each M-variable, if the M-variable is 1 then the corresponding S-variable must also be 1:

$$S_{f,m} \geq M_{w,P,f,m}$$

- *Each position of each word must be in exactly one segment.* For each position in each word, there must be exactly one M-variable that contains that position and whose value is 1. Thus for each position p of each word w we generate the following constraint:

$$\sum_{\substack{f \in \text{features of } w \\ m \in \text{candidate morphemes of } f \\ P \in \text{occurrences of } m \text{ in } w \text{ where } p \in P}} M_{w,P,f,m} = 1.$$

- *In each word, each feature must be mapped to exactly one morpheme.* For each feature f of each word w and feature f of w, exactly one M-variable must be 1:

$$\sum_{\substack{m \in \text{candidate morphemes of } f \\ P \in \text{occurrences of } m \text{ in } w}} M_{w,P,f,m} = 1.$$

Example For the above example, the first rule generates the following constraints, which force an S-variable to 1 when one of its M-variables is 1:

$$\begin{aligned} S_{\text{häst,häst}} &\geq M_{\text{häst,1–4,häst,häst}} \\ S_{\text{Pl,ar}} &\geq M_{\text{häst,5–6,Pl,ar}} \\ S_{\text{Pl,na}} &\geq M_{\text{häst,7–8,Pl,na}} \\ S_{\text{Def,ar}} &\geq M_{\text{häst,5–6,Def,ar}} \\ S_{\text{Def,na}} &\geq M_{\text{häst,7–8,Def,na}} \end{aligned}$$

The second rule generates the following constraints, since letters 1 to 4 can only be covered by *häst*, letters 5 to 6 by *ar* (either as Pl or Def), and letters 7 to 8 by *na*:

$$\begin{aligned} M_{\text{häst,1–4,häst,häst}} &= 1 \\ M_{\text{häst,5–6,Pl,ar}} + M_{\text{häst,5–6,Def,ar}} &= 1 \\ M_{\text{häst,7–8,Pl,na}} + M_{\text{häst,7–8,Def,na}} &= 1 \end{aligned}$$

The third rule generates the following constraints, stating that *häst*, Pl and Def must be marked by exactly one morpheme each:

$$\begin{aligned} M_{\text{häst,5–6,Pl,ar}} + M_{\text{häst,7–8,Pl,na}} &= 1 \\ M_{\text{häst,5–6,Def,ar}} + M_{\text{häst,7–8,Def,na}} &= 1 \\ M_{\text{häst,1–4,häst,häst}} &= 1 \end{aligned}$$

This set of constraints has two solutions:

1. One where Pl is assigned to *ar* and Def to *na*. In this solution, the following variables are 1 and the rest are 0: $S_{\text{häst,häst}}$, $S_{\text{Pl,ar}}$, $S_{\text{Def,na}}$, $M_{\text{häst,1–4,häst,häst}}$, $M_{\text{häst,5–6,Pl,ar}}$, $M_{\text{häst,7–8,Def,na}}$.
2. One where Def is assigned to *ar* and Pl to *na*.

In general, any valid segmentation of the input words is a solution to the constraint problem. To make the constraint solver find the best segmentation, we also supply an objective function to be minimised. In our case, we choose to minimise the total number of morpheme-feature pairs used. To achieve this, we supply the objective function

$$\sum_{\substack{f \in \text{features} \\ m \in \text{candidates of } f}} S_{f,m}.$$

Example Suppose that we add to our earlier example the word *hundars* (dogs'). Its stem is *hund* and its other features are Gen and Pl. We also add *s* to the candidate morphemes for Pl, Def and Gen.

The constraint solver finds the solution that minimises the value of the objective function, which

in this case means assigning *ar* to Pl, *s* to Gen and *na* to Def. The objective function's value is then 3 ($S_{\text{Pl,ar}} + S_{\text{Gen,s}} + S_{\text{Def,na}}$). This is the correct segmentation; the wrong segmentations are rejected because they have more feature-morpheme pairs and thus make the objective function larger.

3.2 Choosing the candidate morphemes: the naive approach

The constraint solver requires as input a set of candidate morphemes for each feature. Since the problem formulation requires that a morpheme or stem of a word in the dictionary must be a subsequence of that word, one option is to simply let the candidate morphemes of a feature f include all subsequences of all words that are annotated with feature f. This guarantees the optimal solution with respect to the given constraints. We have found two main problems with this approach.

1. While it works very well for small inputs (around 20 words), the constraint problem quickly becomes infeasible with larger sets of data, especially if many features are involved.

2. It does not consider the position of the morpheme in the word: the suffix *-s* is a plural marker in English, but the infix *-s-* is not.

We solve Problem 1 with an algorithm that guesses an approximate stem and then refines the guess. The algorithm is described in section 3.4.

To solve Problem 2, we now introduce morpheme patterns, which restrict the way in which morphemes can be applied: in this case, a morpheme that has been observed only as a suffix in the data should only occur to the right of the stem.

3.3 Morpheme patterns

If we do not distinguish prefixes, suffixes and infixes, we can not know whether the word seashells should be segmented as seashell|s or s|eashells or even sea|s|hells. *Morpheme patterns* allow us to make this distinction. A typical morpheme pattern is +s∗, which represents the suffix *s*.

Morpheme patterns act as jigsaw pieces that make sure each morpheme is placed in its appropriate position in relation to the stem. By using morpheme patterns, we restrict the way morphemes can be combined and obtain a more precise segmentation. The purpose of this section is to formalise what + and ∗ in patterns mean, and to extend the segmentation algorithm to respect the meaning of the patterns. Both stems and affixes are described using morpheme patterns, but their semantics are slightly different.

Stem Patterns For stems, a ∗ symbol marks a position where an affix can (but does not have to be) inserted. For example, *m∗n∗* is the stem of *man*, where an *a* can be inserted in the infix position to make it singular, or an *e* to make it plural. In the suffix position, we can have the null morpheme, or add an affix such as *'s*. To accommodate word forms with multiple tokens, such as the German word *fängt an* (begins), with standard form *anfangen*, stems are represented by a set of patterns. The patterns of this set can be combined in any order, with or without a space. {f∗ng∗, an} is thus a stem of both word forms.

Affix Patterns Affix patterns include two special symbols, + and ∗. For an affix pattern to match a word, you must be able to obtain the word by replacing each + and ∗ with an appropriate string. For example, +s∗ matches *dogs'* by replacing + with *dog* and ∗ by an apostrophe. But there are two important restrictions:

- Each + must be replaced by a string that contains some part of the stem.

- Each ∗ must be replaced by a string that *does not* contain any part of the stem.

In effect, the + symbol determines where the stem must be placed in relation to the affix, while the ∗ symbol allows other affixes to be attached in its place. For example, the plural morpheme *+s∗* in English must be placed after (but not necessarily directly after) a stem. Likewise, the genitive morpheme *+'∗* must have the stem entirely on its left side. The two morphemes can be combined into *+s'∗*, and be placed after the stem *horse∗*, to produce the word *horses'*. The morpheme *+ea+* must be placed where it has the stem on both sides, thus making it an infix. Together with the stem pattern *br∗k∗*, it produces the word *break*. Similarly, the morpheme *+o+en∗* together with *br∗k∗* produces the word *broken*. An affix pattern can in theory have any number of +-symbols.

Extra constraints In order to make the constraint solver follow the semantics of stem patterns and affix patterns, we must for each word add extra constraints for the stems and patterns that are incompatible. An affix and stem can both

be chosen for a word only if the affix is in an appropriate position in relation to the stem. Thus, for all words w, and all pairs of M-variables $M_{w,P_1,f_1,m_1}, M_{w,P_2,f_2,m_2}$, we check if m_1 is a stem pattern, m_2 is an affix pattern and their positions P_1 and P_2 are incompatible. If so, we add a constraint that only one of them can be chosen:

$$M_{w,P_1,f_1,m_1} + M_{w,P_2,f_2,m_2} \leq 1$$

For example, supposing we have *+e+* as a candidate morpheme for *plural*, *+s'* as a candidate morpheme of *genitive*, and *bus∗* as a candidate stem of *buses'*, the constraint will not accept *bus|e|s'* as a segmentation, since *+e+* is required to be an infix of the stem.

3.4 The Algorithm

Considering all possible subsequences of all words yields an unmanageable number of variables for the constraint solver. We therefore need to identify the relevant subsequences, which is done in several steps.

1. As a first step, we approximate the stem as the longest common subsequence of the inflected word and the standard form:

standard	inflected	features
woman	**women**	Pl;Nom
baby	**babies'**	Pl;Gen
dog	**dogs'**	Pl;Gen
lorry	**lorry**	Sg;Nom

2. By removing the approximated stems, we simplify the problem to segmenting the non-stem part of each word. The resulting problem includes many duplicates, as well as having shorter strings to segment, making it feasible to naively consider every subsequence as a candidate for each feature listed with the word. The result is as follows.

women	e	→	e | Pl Nom
woman	a	→	a | Sg Nom
babies'	ies'	→	ies | Pl ' Gen
baby	y	→	y | Sg Nom
dogs'	s'	→	s | Pl ' Gen
dog		→	| Sg Nom
lorry		→	| Sg Nom

We simplify the problem further, by automatically giving the null morpheme to features that are shared between the inflected form and the standard form. Since we do not allow any segment to be explained by multiple features, we can assume that the word difference is unrelated to the features that are shared between the two word forms. As an optimisation, when a word is annotated with a single non-shared feature, that feature is automatically paired with the entire non-stem part, and no subsequences are generated.

The approximated stem together with the segmentation of the remainder of the word makes a first approximation of a segmentation of the entire word. However, choosing the longest common subsequence as the stem does not always result in the best segmentation. In our example above, the *-y* is dropped in the stem of *baby*, but is included in the stem of *lorry*. A more consistent choice would be to drop it for *lorry* too, which we achieve in the next step.

3. Taking the morphemes chosen by the constraint solver in step 2, we generate additional stem candidates by removing from each word each possible choice of morpheme for the word's features. For example, in the segmentation to the left, we got *+y∗* as a morpheme of the *Sg* feature. Therefore, for any word with the *Sg* feature that ends in *y*, we generate a candidate stem where the *-y* suffix is dropped.

   For *lorry*, we get the candidate stem *lorr∗* in addition to the approximated stem *lorry* from step 1. For *baby*, in addition to the suffix *+y*, we also consider the infix *+a+* from *wom∗n* in step 2 as a possibility for the singular morpheme. The candidate stems thus become *bab∗* and *b∗by*.

4. Using the morphemes computed in step 1 and the stem candidates of step 3, we re-run the constraint solver and let it find the optimal choice of stems and morphemes. The chosen stems and morphemes are decoded from the 1-valued S-variables, while the segmentation can be decoded from the 1-valued M-variables.

4 Finding morphological rules

We have now discovered which morphemes mark each features. This section shows how to find in-

flections that change one morpheme to another, and morphemes that mark a combination of features.

4.1 Function features

In order to detect inflection rules that involve *replacing* one morpheme with another, such as the change of the suffix *+y∗* into *+ie∗* in English plural, we introduce the concept of *function features*. Without function features, we might find all of *+s∗*, *+es∗* and *+ies∗* as English plural morphemes. By adding function features, we can specify that *+s∗* is generally used as the pluralisation morpheme, and when *+ies∗* and *+es∗* are used instead.

Function features are a special synthetic feature automatically added to some input words. They are added once we have segmented the input, and only for words where the standard and inflected form have the same stem. Let us take as an example the word *babies*, its standard form *baby*, and its stem *bab∗*. We first remove the stem from the standard form to get *+y∗*. Our idea is that, since *+y∗* is a suffix, we will add the feature *From_Suffix_y* to the set of features of *babies*. We process the whole input this way, and then re-run the segmentation. Assuming there are several words in the input that share the same paradigm, the constraint solver will map *+s∗* to *Plural* (because the pair (*+s∗*, *Plural*) is commonly occurring and shared by other words), and *+ie∗* to the *From_Suffix_y* feature. This segmentation captures the fact that words ending in *y* often change to *ie* in plural. We can picture this process as: $\text{bab(y} \rightarrow \text{ie)} \mid_{\text{Pl}} \text{s}$.

In the same way, the word-lemma pairs *men - man, wives - wife and mice - mouse* result in the stem changes *+a+* $\rightarrow$ *+e+*, *+f+* $\rightarrow$ *+v+* and *+ous+* $\rightarrow$ *+ic+*, after we synthesise the respective function features *From_Infix_a, From_Infix_f* and *From_Infix_ous*.

An inflection rule may also be specific to a part of the stem, such as the doubling of the letter *g* in *big - bigger*. To cover cases involving the last or first letter of the stem, we synthesise the feature *AddTo_Suffix_x*, where x is the last letter of the stem, and *AddTo_Prefix_x*, where x is the first letter of the stem. As an example, *bigger*, with the stem *big∗*, is given the additional feature *AddTo_Suffix_g*. The morpheme *+er∗* is likely to be mapped to the comparative feature (due to its commonness in words in comparative form), while the remaining *+g∗* is mapped to the *AddTo_Suffix_g* feature:

$\text{bi(g} \rightarrow \text{gg)} \mid_{\text{Comp}} \text{er}$

We also capture the phenomenon where extra letters are inserted when adding an affix, such as the insertion of an extra *g* between the past tense marker *ge* and a stem beginning with the letter *e* in German. To do so, we identify the first and last letters of the stem and add them as synthesised features. In this case, this results in the extra feature *AddTo_Prefix_e*, the segmentation of *gegessen* (eaten) becomes $\text{ge}\mid_{\text{Past}}\text{(e} \rightarrow \text{ge)ss}\mid_{\text{Past}}\text{en}$.

This algorithm is *not* language-specific: it works for any language where the concepts of prefix, infix, suffix, first and final letter affect inflection.

4.2 Morphemes with multiple features

Sometimes, a morpheme can be linked to more than one feature. For example, *+na∗* in Swedish is a morpheme of definite form, but it occurs only in plural. To find such links we post-process the result returned by the constraint solver. For each morpheme *m* of each feature *f*, we collect all words whose segmentation uses morpheme *m* for feature *f*. The intersection of all features of all such word entries reveals what features always occur together with the combination of m and f.

5 Experimental Results

We evaluated our tool on four different problems: English nouns, Swedish nouns, Dutch verbs, and the SIGMORPHON 2018 shared task of morphological reinflection (Cotterell et al., 2018).

English nouns We tested our tool on 400 randomly selected English nouns from the GF resource grammar library (Ranta, 2009). The tool took 3.9s to run. Fig. 1 shows the morphemes chosen before and after the addition of function features.

From the test results, we randomly select 20 words to demonstrate their segmentations. The first segmentation, based on approximated stems (step 3 of the algorithm) is presented in Fig. 2.

Step 4 of the algorithm changes just one entry: the suffix *y* has been dropped from the stem of *sky*. None of the 20 word entries involves a function feature that was assigned to a non-null morpheme.

Combined morphemes Fig. 3 lists the morphemes with multiple features, as described in section 4.2. The list of combined features nicely shows many of the inflection rules appearing in the data.

Reducing the test data We repeat the experiment after reducing the test data to include only

First approximation of morphemes

feature	morphemes
Gen	∗ +'s∗ +'∗ +e+'s∗ +v+'∗
Nom	∗
Pl	∗ +s∗ +ies∗ +es∗ +e+ +ren∗ +v+s∗ +ic+
Sg	∗ +y∗ +a+ +f∗ +f+ +oo+ +ous+

Morphemes, including function features

feature	morphemes
Gen	∗ +s' +'s
Nom	∗ +ous+
Pl	∗ +s∗
Sg	∗ +a+ +oo+ +y∗ +f∗ +f+
Infix_a	∗ +e+
Infix_f	∗ +v+
Infix_oo	∗ +ee+ +oo+
Infix_ous	∗ +ic+
Suffix_f	∗ +ve∗ +f*
Suffix_y	∗ +ie∗ +y*
AddToSuffix_d	∗ +ren∗
AddToSuffix_h	∗ +e∗

Figure 1: Chosen morphemes after each step

the 20 randomly selected words. After step 2, the segmentation based on 20 words is identical to the segmentation based on 400 words, with just one exception; the stem of *country* includes the suffix *y*. After step 3 and step 4, the segmentation is identical to the one based on 400 words.

Swedish nouns and Dutch verbs We also tested our method on a set of Swedish nouns and Dutch verbs. For Swedish nouns, our method works very well. The precision is 100% based on a set of 50 words, and 94% based on a set of 250 words. The erroneous results on the bigger data were because the algorithm noticed a vowel change in certain words and applied it universally, causing for example the stem of *grad* (degree) to wrongly become gr∗d∗ instead of grad∗, because of other words in which *a* becomes *ä* in the plural. For Dutch verbs, the precision was 80%, based on 50 words, and 74% based on 250 words. The errors made were similar to those on the Swedish test data.

Comparison with earlier work We compared our results on Swedish with those of Silfverberg and Hulden (2017), although they do not use the same dataset (in particular, ours only includes nouns). Our precision of 94% far exceeds their

	word	stem	morphemes
1	rivers'	river	+s'/Gen,Pl
2	breasts	breast	+s/Pl /Nom
3	river's	river	+'s/Gen /Sg
4	windows'	window	+s'/Gen,Pl
5	television's	television	+'s/Gen /Sg
6	country	countr+	+y/Sg /Nom
7	languages'	language	+s'/Gen,Pl
8	fire	fire	/Sg,Nom
9	number	number	/Sg,Nom
10	ceiling	ceiling	/Sg,Nom
11	question's	question	+'s/Gen /Sg
12	song	song	/Sg,Nom
13	airplane's	airplane	+'s/Gen /Sg
14	doors'	door	+s'/Gen,Pl
15	fires	fire	+s/Pl /Nom
16	water	water	/Sg,Nom
17	arts'	art	+s'/Gen,Pl
18	flowers'	flower	+s'/Gen,Pl
19	sky's	sky	+'s/Gen /Sg
20	ear	ear	/Sg,Nom

Figure 2: The segmentation of the 20 test words, based on the test data of 400 words

feature		morpheme	combines with
+a+	→	+e+	Pl
(policeman → policemen and 1 other(s))			
+f+	→	+v+	Pl
(wife → wives)			
+oo+	→	+ee+	Pl,Gen
(foot → feet's)			
+ous+	→	+ic+	Pl,Nom
(louse → lice)			
Pl		+s'∗	Gen
(doctor → doctors' and 91 other(s))			
+d	→	+d∗ren	Pl
(child → children)			
+h	→	+h∗e	Pl
(church → churches and 1 other(s))			
+f	→	+ve∗	Pl,Gen
(leaf → leaves')			
+y	→	+ie∗	Pl
(country → countries and 7 other(s))			

Figure 3: The combined features, where the feature of column 1 and morpheme of column 2 occur only in combination with the features of column 3

precision of 62%.[3] To find out why, we looked into what sort of errors both tools made. As mentioned above, our tool made one class of errors, inferring a vowel change where none was needed, but produced a plausible segmentation.

Their tool found many implausible segmentations; for example, *inkomst* (income) was sometimes segmented into i|nkomst, and *pension* into p|ension. Furthermore, it segmented words inconsistently: some occurrences of *inkomst* were (correctly) left as one segment. This means that the tool has not found the *simplest* explanation of the input data: its explanation requires more morphemes than necessary, such as *i*, *p* and *nkomst*. We avoid this problem since the constraint solver guarantees to find the globally minimal solution to the input constraints.

Secondly, their tool does not restrict where in a word a morpheme can occur. For example, the letter *a* can mark common nouns, such as *flicka* (girl). It only occurs as a suffix, but their tool uses it as a prefix to segment *arbetsinkomst* (income from work) into a|rbetsinkomst. By distinguishing different kinds of affixes, we avoid this problem.

Morphological Reinflection We use an adapted version of our tool to solve the SIGMORPHON 2018 shared task of morphological (type level) reinflection (Cotterell et al., 2018). Given a lemma and set of morphological features, the task is to generate a target inflected form. For example, given the source form *release* and target features PTCP and PRS, the task is to predict the target form *releasing*.

Our approach requires a set of labelled training data, which we segment to obtain a list of affixes and their associated features. To predict the target inflected form of a word, we: 1) find the stem of the word, 2) find a word in the training data whose features match the target features, and 3) replace the stem of that word with that of the input word.

In more detail, in step 1, we check if the word contains any affixes that are associated with a feature belonging to the lemma. We remove any such affix from the word. There may be a choice of affixes for each feature so this results in a set of candidate stems. When considering a candidate stem, we also add the appropriate function features to the target feature list; for example, if the stem drops a suffix *suff* from the source form, we add the feature *From_Suffix_suff*. In step 2, for each candidate stem, we collect the entries of the training data that match the target features (including the function features collected in step 1). Out of those, we pick the word whose stem best matches the source form. In step 3, we take this word, and replace its stem with the stem of the input word.

Language	**Our system**	**Mean**	**Best**
Arabic	25.6	14.77	45.2
	(2.95)	(6.63)	(1.77)
Galician	49.0	31.93	61.1
	(1.42)	(2.4)	(0.72)
Greek	27.9	15.76	32.3
	(3.02)	(4.89)	(1.83)
Karelian	32.0	47.93	94.0
	(1.4)	(1.53)	(0.1)
Russian	43.8	26.86	53.5
	(1.41)	(3.64)	(1.07)
Sanskrit	43.9	25.76	58.0
	(1.55)	(2.99)	(0.93)
Slovene	35.9	24.80	58.0
	(1.15)	(2.7)	(0.73)
Tatar	64.0	48.89	90.0
	(0.44)	(2.15)	(0.14)
Telugu	66.0	67.41	96.0
	(0.98)	(1.29)	(0.06)
West Frisian	43.0	32.52	56.0
	(1.86)	(1.85)	(1.01)

Figure 4: Results of reinflection. The first line gives the average accuracy and the second line the average Levenshtein distance from the right answer.

We evaluate our system on the *low* training data, the smallest of the three available sizes, which consists of 100 words for each of the 103 languages. The data includes a mixture of nouns, adjectives and verbs. 17 of the languages were excluded from the evaluation, because they involved a large number of features, resulting in a too long execution time. On the remaining 86 languages, our approach performs, with a few exceptions, in the better half, and often in the better third of the 27 submitted systems. For English, Czech, Greek, Livonian and Romanian, the accuracy is within 5% of the accuracy of the system with the highest score. Figure 4 shows the accuracy and Levenshtein distance of our system on a sample of languages, and the mean and best values of the systems that took part in the shared task. Our reinflection algorithm is very simple, but still competes with state-of-the-art sys-

[3] 62% is the figure for *unlabelled* morphemes. The figures given in the paper for *labelled* morphemes are unfortunately erroneous.

tems, indicating that the underlying morphological analysis provided by our tool is of good quality.

6 Conclusion and Future Work

We have presented a method for morphological segmentation based on constraint solving. Our evaluation shows that it works well even given little training data, producing almost the same segmentation from 20 English words as it does from 400 words. It produces a morphological analysis precise enough that a simple reinflection algorithm built on it can compete with state-of-the-art systems. The reasons for this good precision are (a) the pattern semantics, which allows the solver to make precise distinctions between different kinds of morphemes, such as infix and suffix; (b) the use of function features to express replacement rules.

The paper also demonstrates that constraint solving is a useful alternative to machine learning for segmentation, particularly for low-resource languages where one must make the most of a small set of data. We hope that our paper spurs further research in this direction.There are many possible refinements to our technique and some of our ideas for future work are listed below.

More refined semantics for morphemes With a more refined semantics of morphemes, we can guide the constraint solver to pick a segmentation that follows the observed data in a more precise way. This can be done by restricting how a morpheme can be placed in relation to other morphemes. For example, the definite morpheme *+na∗* in Swedish always follows a plural morpheme (*+er∗*, *+or∗* or *+ar∗*), and never occurs directly after a stem. The morpheme semantics could be improved to allow these kind of restrictions, and the algorithm refined to automatically infer them from the data.

Improved function features Currently, we consider only the first and last letter of the stem for stem additions, as described in section 4. We are currently investigating generalising this idea by using the constraint solver to find the relevant segment of the stem. This would allow us to detect a wider range of morphological changes.

As an example, suppose we would like to find out which kinds of English words form their plural with *+es*. We could take all such words that occur in the input, and give the following segmentation problem to the constraint solver:

standard	**features**
bus	bus; Suffix_es
dress	dress; Suffix_es
box	box; Suffix_es
suffix	suffix; Suffix_es

The solution returned indicates which letter patterns are associated with the plural form *+es*:

bu|s (Suffix_es)
dres|s (Suffix_es)
bo|x (Suffix_es)
suffi|x (Suffix_es)

We could then introduce new features *AddTo_Suffix_s* and *AddTo_Suffix_x* to the original problem, whereupon the algorithm of Section 4 would find the correct function features. This method would be able to invent function features from an arbitrary substring of the given words.

Tweaking the objective function The objective function can be weighted to give higher or lower cost to stems or morphemes related to specific kinds of features. For example, by multiplying each term $S_{f,m}$ in the objective function by the length of m, we would recover the Minimum Description Length principle. It is left as future work to investigate how the choice of cost function affects the result in different settings.

Distinguishing between rules and exceptions Once a morpheme is used in a segmentation, the algorithm is sometimes too eager to use the same morpheme elsewhere. This means that adding more data sometimes leads to worse results, and errors in the input can cause unrelated words to be segmented wrongly. We plan to investigate using statistical methods to distinguish between morphological rules and exceptions; exceptions should not be applied everywhere, but rules should.

Scalability For most languages our tool works comfortably up to a thousand words or more, but for languages with many morphosyntactic features (such as Basque) it can struggle to deal with a hundred words. We would like to see if, by tackling features in smaller groups, it is possible to scale the approach to large inputs.

References

Malin Ahlberg, Markus Forsberg, and Mans Hulden. 2014. Semi-supervised learning of morphological paradigms and lexicons. In *Proceedings of the 14th*

Conference of the European Chapter of the Association for Computational Linguistics, Gothenburg, Sweden 26–30 April 2014, pages 569–578.

Jonathan Berant, Ido Dagan, and Jacob Goldberger. 2011. Global learning of typed entailment rules. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 610–619, Portland, Oregon, USA. Association for Computational Linguistics.

James Clarke and Mirella Lapata. 2008. Global inference for sentence compression: An integer linear programming approach. *Journal of Artificial Intelligence Research*, 31:399–429.

Ryan Cotterell, Christo Kirov, John Sylak-Glassman, Géraldine Walther, Ekaterina Vylomova, Arya D McCarthy, Katharina Kann, Sebastian Mielke, Garrett Nicolai, Miikka Silfverberg, et al. 2018. The conll–sigmorphon 2018 shared task: Universal morphological reinflection. *arXiv preprint arXiv:1810.07125*.

Mathias Creutz and Krista Lagus. 2007. Unsupervised models for morpheme segmentation and morphology learning. *ACM Trans. Speech Lang. Process.*, 4(1):3:1–3:34.

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

Spence Green and John DeNero. 2012. A class-based agreement model for generating accurately inflected translations. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers - Volume 1*, ACL '12, pages 146–155, Stroudsburg, PA, USA. Association for Computational Linguistics.

Stig-Arne Grönroos, Sami Virpioja, Peter Smit, and Mikko Kurimo. 2014. Morfessor flatcat: An hmm-based method for unsupervised and semi-supervised learning of morphology. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 1177–1185, Dublin, Ireland. Dublin City University and Association for Computational Linguistics.

Harald Hammarström and Lars Borin. 2011. Unsupervised learning of morphology. *Computational Linguistics*, 37(2):309–350.

Zellig S. Harris. 1955. From phoneme to morpheme. *Language*, 31(2):190–222.

IBM ILOG. 2009. *IBM ILOG CPLEX V12.1: User's manual for CPLEX*.

Mark Johnson. 2008. Using adaptor grammars to identify synergies in the unsupervised acquisition of linguistic structure. In *Proceedings of ACL-08: HLT*, pages 398–406, Columbus, Ohio. Association for Computational Linguistics.

Oskar Kohonen, Sami Virpioja, and Krista Lagus. 2010. 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. Association for Computational Linguistics.

Jiaming Luo, Karthik Narasimhan, and Regina Barzilay. 2017. Unsupervised learning of morphological forests. *Transactions of the Association for Computational Linguistics*, 5:353–364.

Christian Monson, Jaime Carbonell, Alon Lavie, and Lori Levin. 2007. Paramor: Minimally supervised induction of paradigm structure and morphological analysis. In *Proceedings of Ninth Meeting of the ACL Special Interest Group in Computational Morphology and Phonology*, pages 117–125, Prague, Czech Republic. Association for Computational Linguistics.

Joakim Nivre, Željko Agić, Lars Ahrenberg, Maria Jesus Aranzabe, Masayuki Asahara, and Aitziber et al. Atutxa. 2017. Universal dependencies 2.0. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Hoifung Poon, Colin Cherry, and Kristina Toutanova. 2009. Unsupervised morphological segmentation with log-linear models. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, NAACL '09, pages 209–217, Stroudsburg, PA, USA. Association for Computational Linguistics.

Karthik Rajagopal Narasimhan, Damianos Karakos, Richard Schwartz, Stavros Tsakalidis, and Regina Barzilay. 2014. Morphological segmentation for keyword spotting.

Aarne Ranta. 2009. The gf resource grammar library.

Dan Roth and Wen-tau Yih. 2005. Integer linear programming inference for conditional random fields. In *Proceedings of the 22Nd International Conference on Machine Learning*, ICML '05, pages 736–743, New York, NY, USA. ACM.

Teemu Ruokolainen, Oskar Kohonen, Kairit Sirts, Stig-Arne Grönroos, Mikko Kurimo, and Sami Virpioja. 2016. A comparative study of minimally supervised morphological segmentation. *Comput. Linguist.*, 42(1):91–120.

Teemu Ruokolainen, Oskar Kohonen, Sami Virpioja, and Mikko Kurimo. 2013. Supervised morphological segmentation in a low-resource learning setting using conditional random fields. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 29–37, Sofia, Bulgaria. Association for Computational Linguistics.

Tarek Sakakini, Suma Bhat, and Pramod Viswanath. 2017. Morse: Semantic-ally drive-n morpheme segment-er. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 552–561, Vancouver, Canada. Association for Computational Linguistics.

Patrick Schone and Daniel Jurafsky. 2000. Knowledge-free induction of morphology using latent semantic analysis. In *Fourth Conference on Computational Natural Language Learning and the Second Learning Language in Logic Workshop*.

Miikka Silfverberg and Mans Hulden. 2017. Automatic morpheme segmentation and labeling in universal dependencies resources. In *Proceedings of the NoDaLiDa Workshop on Universal Dependencies, UDW@NoDaLiDa 2017, Gothenburg, Sweden, May 22, 2017*, pages 140–145. Association for Computational Linguistics.

Kairit Sirts and Sharon Goldwater. 2013. Minimally-supervised morphological segmentation using adaptor grammars. *Transactions of the Association for Computational Linguistics*, 1:255–266.

Radu Soricut and Franz Och. 2015. Unsupervised morphology induction using word embeddings. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1627–1637, Denver, Colorado. Association for Computational Linguistics.

Ville T. Turunen and Mikko Kurimo. 2008. Speech retrieval from unsegmented finnish audio using statistical morpheme-like units for segmentation, recognition, and retrieval. *ACM Trans. Speech Lang. Process.*, 8(1):1:1–1:25.

Ivan Vulić, Nikola Mrkšić, Roi Reichart, Diarmuid Ó Séaghdha, Steve Young, and Anna Korhonen. 2017. Morph-fitting: Fine-tuning word vector spaces with simple language-specific rules. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 56–68, Vancouver, Canada. Association for Computational Linguistics.