Algorithms for the Verification of the Semantic Relation Between a Compound and a Given Lexeme

Gudrun Kellner
Vienna University of Technology, Inst. of Software Technology and Interactive Systems
+43(1) 58801-18819, Austria
gudrun.kellner@tuwien.ac.at

Johannes Grünauer
Vienna University of Technology, Inst. of Software Technology and Interactive Systems
+43(1) 58801-18819, Austria
johannes.gruenauer@tuwien.ac.at

ABSTRACT

Text mining on a lexical basis is quite well developed for the English language. In compounding languages, however, lexicalized words are often a combination of two or more semantic units. New words can be built easily by concatenating existing ones, without putting any white spaces in between.
That poses a problem to existing search algorithms: Such compounds could be of high interest for a search request, but how can be examined whether a compound comprises a given lexeme? A string match can be considered as an indication, but does not prove semantic relation. The same problem is faced when using lexicon based approaches where signal words are defined as lexemes only and need to be identified in all forms of appearance, and hence also as component of a compound. This paper explores the characteristics of compounds and their constituent elements for German, and compares seven algorithms with regard to runtime and error rates. The results of this study are relevant to query analysis and term weighting approaches in information retrieval system design.

Categories and Subject Descriptors

I.2.7 Natural Language Processing

General Terms

Algorithms, Measurement, Performance, Reliability, Experimentation.

Keywords

Information Retrieval, Compound, Stemming, Semantic relation.

1. INTRODUCTION

Especially for compounding languages, such as German, Dutch, Danish, Norwegian, Swedish, Russian or Finnish, compound handling is still a challenge – and even more when a word should be analyzed by its semantic components. The subject of our work is the development, implementation and comparison of different algorithms that are able to decide whether a given German lexeme that occurs within a word is a complete semantic component of this word or just a part of another semantic component haphazardly containing the same letters. Searching e.g. for the semantic component *blau* [blue], one would want to have words like *kobaltblau* [cobalt blue] or *Blauhelm* [blue helmet] with a semantic relation to the word *blau* evaluated positively, whereas words like *Ablaufsteuerung* [sequence control system] or *halblaut* [in a low voice] with no relation to the word *blau* should be evaluated negatively.

The scope of this work is therefore different from typical stemming or decompounding approaches where the word components are unknown and need to be identified first. Even though one can use the same methods, the discussed problem is more concrete and can thus be solved faster and more efficiently with adequate tools. The aim of this paper is to develop such algorithms that are easy to implement and to use, and to test them for their efficiency.

Such an algorithm can, e.g., be used to specify an additional filtering level of search results by identifying those words that do not only include the given lexeme as a string, but are also semantically linked to it. The result is a new subset R of relevant results within the search results S from a collection of words W. The relations between those sets are depicted in Figure 1. Furthermore, such an approach could be very useful for lexicon based methods categorizing text with the help of signal words that can occur in various appearance forms, such as affective [3] or sensory [16] vocabulary.

To specify this set of relevant results R, we developed different algorithms based on word formation rules, some of them also using a dictionary and affix lists. The algorithms cover different approaches, ranging from phonetic to elaborate linguistic methods. For testing those algorithms, we chose (out of a list of several hundred lexemes) a set of 16 lexemes that induce different treatment in our algorithms and evaluated 1657 words containing one of these lexemes. We introduce these seven algorithms and compare their results, with regard to runtime and error rate.

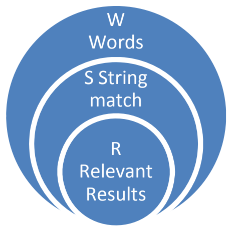


Figure 1. Subsets of search results

The paper is organized as follows: Section 2 provides some linguistic background and explores related research. Section 3 and 4 describe the general structure of all algorithms and the specific structure of each algorithm. Section 5 explains the methods we used to compare those algorithms. Section 6 presents the results, including some discussion. Section 7 provides a conclusion and an outlook to further research.

2. BACKGROUND AND RELATED RESEARCH

Related research comes from the fields of morphology, phonology, and computational linguistics as well as information retrieval. First, we investigate the components of word formation and their structure in compounding languages, using German as example language. Second, we focus on stemming and compound splitting and take a look at methods and strategies used in this field.

2.1 Components of Word Formation

The structure of German words can be broken down to the following constituents:

- Lexemes are abstract word forms that express semantic content without giving grammatical information. We use the expression 'lexeme' as described by [23] as archilexeme, which includes all distinctive shared attributes of two semantically identic, but grammatically differently realized words; e.g. the allomorphic stems from the word *laufen* [to run] in its realizations *lauf*, *läuf* [run], and *lief* [ran].
- Affixes are morphemes that are attached to a word or its stem in order to form a new word. The most common forms are prefixes that are attached before, and suffixes that are attached after a word or its stem. Affixes can express grammatical information, but might also change the semantic content of a word.
- Flexion endings are put at the end of a word and express grammatical information. They can be split up into two groups. Declination endings are used for nouns and adjectives whereas conjugation endings are used for verbs.
- Word joints are linking morphemes that might occur within compounds and exist in paradigmatic (e.g. genitive or plural inflection) or nonparadigmatic form.

The German language is productive, which means that new words can be built by following existing morphemic patterns. The most important types of word formation are derivation, conversion and compounding. Derivation describes word formation with the help of affixes, especially prefixes and suffixes, and the use of Ablaut/Umlaut as markers for implicit derivation. Conversion or syntactical transposition describes the change of part of speech without any other morphological markers. Compounding is a process that combines at least two semantic components within one word. The place where different semantic components meet is called inner word border. It can be accompanied by a word joint.

All those morphemic patterns can be described by word formation models as the combination of one or more lexemes, affixes, word joints, and flexion endings. But description by morphemic components is not the only way to describe the structure of a word. Phonotactic models describe words as suites of consonant and/or vowel clusters and use statistical methods and rules in order to identify possible consonant/vowel clusters for a given language. The only word formation type that cannot be described that way is blending, that is the formation of words by building abbreviations; therefore we will not take that phenomenon into account.

2.2 Stemming and Compound Splitting

Stemming is a procedure that conflates word forms to a common stem. For example, the word forms *runner*, *runs*, and *running* are all reduced to *run* by a stemmer. Stemmers usually interpret white spaces and punctuation as word borders. This is a powerful method for non-agglutinative languages like English, where the components of ad hoc compounds tend to stay visibly separated by white spaces or hyphens, e.g. *compound verb idioms* or *long-lasting*, whereas one-word compounds are usually lexicalized. However, stemming does only cut away flection endings and some suffixes, but does not consider the inner structure of a word, which is problematic for long compounds such as the German word *Donaudampfschiffahrtsgesellschaftskapitäne* [captains of Danube steamboat electrical services] that a stemmer could only reduce to its singular *...-kapitän*. At this point, compound splitting comes into use. Compound splitting analyzes a compound for its internal structure and splits a compound into its components [20].

Different strategies are pursued in order perform stemming, only some of them analyze the structure of compounds. Whereas brute force algorithms work with complete lookup tables that contain relations between root forms and inflected forms for all word forms (which is quite unrealistic for compounding languages such as German), suffix stripping algorithms make use of lists of suffixes that can be cut away in order to perform stemming [18, 19]. Morphologic algorithms work with morphologic description of words [17]. Lemmatization algorithms use a more detailed grammar description as they first determine the part of speech of a word and then apply different normalization rules for each part of speech [13]. Stochastic stemming algorithms are based on frequency and probability: with the help of unsupervised learning, huge corpora can be used as a reference in order to find similar solutions for a given word that needs to be stemmed or split up into its components [12]. A similar concept is proposed by [1] who use an annotated corpus as training set. In their overview paper on decompounding in German, [5] analyze some more approaches like n-gram-based information retrieval and the use of well-engineered and mature stemmers like the NIST stemmer or the spider stemmer.

One of the biggest challenges of natural language processing is ambiguity, especially on a morphological level [4]. Both stemming and decompounding should be executed in a well-balanced extent. In the case of 'overstemming', too much is cut away by a stemmer; if too little is cut away, one can speak of 'understemming'. For compound splitting, those effects are called 'aggressive' respectively 'conservative' decompounding [5].

Stemming and decompounding are used for processing user queries [2] as well as in the field of machine translation [21].

3. GENERAL STRUCTURE OF THE ALGORITHMS

Even though our algorithms make use of different strategies, all algorithms have the same structure. Each algorithm is a realization of the parameterization as modeled in Figure 2. The input elements are a given string (a lexeme) and a word that includes that string. The verification process analyzes the semantic relation between the word and the given string. The output is Boolean (true/false) and consists of a decision about that semantic relation of the two input parameters.

We developed seven different algorithms. Besides the naïve search algorithm, all are based on word formation rules, some of them also using a dictionary and affix lists. They all have the same structure and use the same interface.

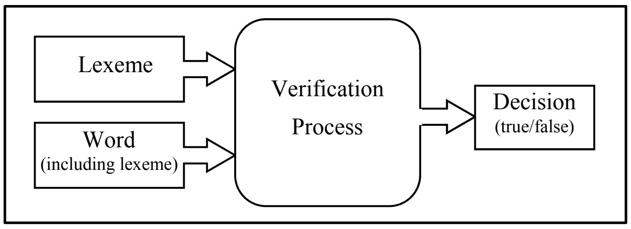


Figure 2. Parameterization of the algorithms

As an input, all algorithms need a given lexeme and a word that includes that lexeme. Some algorithms need additional information concerning the possible part of speech (further addressed as PoS) realizations of the lexeme. Those options are coded binary considering that one lexeme can have several possible PoS realizations (e.g., *ärger* [trouble] can be realized as an adjective, a noun or a verb). We only took the three most current PoS realizations of the German language into account, namely adjectives, nouns and verbs. Table 1 shows the different PoS realization types and their codification.

Table 1. Part of speech (PoS) realization types

Code	Possible PoS realization
1	adjective
2	noun
3	adjective, noun
4	verb
5	adjective, verb
6	noun, verb
7	adjective, noun, verb

3.1 Problem description

The starting point for each algorithm at the moment of its function call is illustrated in Figure 3. It shows two compounds containing the string *röt* that can be produced by vowel gradation on the lexeme *rot* [red].

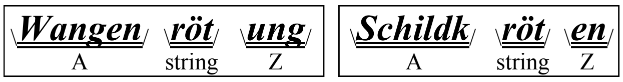


Figure 3. Decision problem

As the input arguments are a lexeme and a word that contains that lexeme, we know the start position of the lexeme in the word and what is before (= part "A") and after (= part "Z") that string. The mission of each algorithm is now to decide whether A and Z are valid letter combinations, built on the respective word grammar model. Looking at our examples, *Wangenrötung* [cheek reddening] should be evaluated positively, whereas *Schildkröten* [turtles] should be evaluated negatively.

3.2 Resources

The algorithms can use resources that are stored in a database. It contains

- a dictionary of all German word forms: Each lexical entry is listed several times in all possible declination and conjugation versions.
- a list of the most frequently used affixes of the German language. Each entry contains the following information:
  - the string of the affix
  - the PoS it can be used with (see the codes in Table 1)
  - the PoS it produces (see the codes in Table 1)
  - the affix type (word joint = 1, prefix = 2, suffix = 4)
  - and a specification on the suffix type (not a suffix = 0, suffix = 1, ending = 2, suffix or ending = 3).

  The entry for the suffix *lich*, e.g., is stored as *lich, 7, 1, 4, 1.*

4. THE ALGORITHMS

In this section, we introduce our algorithms. The algorithms cover different approaches, ranging from phonetic to elaborate linguistic methods. While algorithm 0 only performs naïve string search, algorithm 1 and 2 are realizations of different morphologic word description models. Algorithm 3 is a lemmatization algorithm. Algorithm 4 uses suffix stripping; algorithm 5 is based on phonotactic rules. Algorithm 6 is a realization of a stochastic stemming algorithm that uses unsupervised learning. If the respective method does not cover the analysis of important word components, the method is completed by operational sequences defined by previous algorithms; e.g. suffix stripping only covers suffix handling and has no rules for prefix handling, which are taken from algorithm 1.

4.1 Algorithm 0

Algorithm 0 is a naïve search algorithm that tests only whether a word contains the letters of the given lexeme or not. As soon as that is the case, the request is evaluated positively. It is used as a baseline.

4.2 Algorithm 1

Algorithm 1 uses an affix list and a dictionary and tests if the letters before and after the given lexeme might be split up into a composition of affixes and dictionary entries. If this is the case, the word is evaluated positively.

4.3 Algorithm 2

Algorithm 2 uses a better word formation model than algorithm 1 and is based on [15].

Kellner differentiates two sorts of suffixes: "preendings" that occur directly after a root word and "endings" that are a result of conjugation or declination and indicate the end of a word. Any word can hence be described as a regular expression[1] with the following structure:

((prefix) (root word){1} (preending)* (word joint)?)?*
(prefix) (root word){1} (preending)* (ending)?*

The structure of our example *Wangenrötung* can thus be analyzed as *(root word)(word joint)(root word)(preending)*.

For comparison reasons, our realization is a simplified version and does not handle exceptions. The rest works in the same way as algorithm 1.

[1] Brackets define the application area of a quantifier. Quantifiers define how often an element may be repeated: * = unlimited times, ? = 0 or 1 times. Other numbers of repetition are numbers within curly braces, e.g. {1} = exactly once, {3-5} = 3, 4 or 5 times, etc.

4.4 Algorithm 3

Algorithm 3 uses a similar system to algorithm 2, but is extended by a rule set deciding which PoS realization type may be combined with which affixes. E.g., the suffix *bar* [-able] can only be combined with PoS realization type 4 (verbs) and produces an adjective. In contrast to previous algorithms, this one uses all the information stored in the affix lists (structure to be found in Section 3.2) and needs, as additional input, the PoS realization type of the given string (to be found in Table 1).

4.5 Algorithm 4

Algorithm 4 is based on Porter's stemming algorithm [18], respectively on its adaptation for German [19] which describes a word as a sequence of consonant and vowel clusters. Porter's stemming algorithm does without complex description of word grammar. It simply divides words into vowel (V) and consonant (C) clusters which are defined as a suite of at least one letter of the same class (either vowels or consonants). Each word can hence be reduced to a regular expression of the following structure:

(C)? (VC){m} (V)?

The word *Schildkröte* [turtle] can thus be notated as *CVCVCV (m=2)* and *Pfau* [peacock] as *CV (m=0)*. From that structure, the algorithm cuts away predefined suffixes under given circumstances. The result is a (somehow) stemmed word without conjugation and declination endings. Porter's algorithm is not very aggressive due to overstemming problems in the context of information retrieval and it only describes suffix stripping.

For comparison reasons, we combine Porter's method with our previous strategies by first executing Porter's suffix stripping and then looking up the remaining letters in our suffix lists and the dictionary.

4.6 Algorithm 5

Algorithm 5 is based on phonotactic rules and is the only one not using a dictionary.

Phonotactics is a subfield of phonetics that describes the permissible combinations of phonemes for a given language [8]. Consonantal phonemes can be combined to consonant clusters considering phonotactic rules. In German, e.g., the phoneme *g* can be followed by an *l* like in *glauben* [believe], but not by a *p*. Still, such a consonant combination can occur at the inner word borders of a compound, like in *Bergpass* [mountain pass] or *Fertigprodukt* [convenience product].

The algorithm works analogously to the first algorithms concerning affixes, but instead of the dictionary lookup it tests whether the last consonants before and the first consonants after the given string are in the list of known consonant clusters for the German language. If both parts are found, the word is evaluated positively.

4.7 Algorithm 6

Algorithm 6 uses the method of supervised learning. All words are described as a composition of up to three word components before and up to three word components after the signal word. The algorithm needs to be trained on a set of words where those components are explicitly defined and works with a decision tree that looks for the word in the training set that corresponds in structure best to the word in processing.

In order to find a common structure for all words, all words from the training corpus need to be annotated according to the schema that can be found in Figure 4.

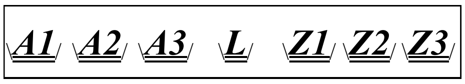


Figure 4. Word fragmentation

Each word fragment is classified and assigned a value according to Table 2.

Table 2. Word fragment classification

Fragment	Classification
L (Given Lexeme)	1 = adjective
	2 = noun
	4 = verb
A1-3 and Z 1-3	0 = empty
	1 = word joint
	2 = prefix
	3 = dictionary match
	4 = suffix
	5 = none of above

Shorter words are surrounded by word fragments with the value 0. The value 5 is assigned to word fragments that cannot be identified as an affix or a dictionary entry and leads to a negative evaluation.

5. IMPLEMENTATION

In this section, we describe how we implemented the algorithms presented previously, looking at structure, input and output, and the database and its components.

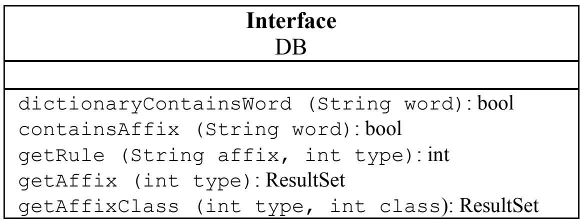
As already described on an abstract level in section 3, all algorithms are built with the same structure. As an input, all algorithms need a given lexeme and a word that includes that lexeme. Valid calls would, e.g., be `eval(Kind, Kindergarten)` [child, kindergarden] or `eval(rot, Roggenbrot)` [red, rye bread]. The correct output would be `true` for the first and `false` for the second call.

Valid calls for the algorithms that need additional information concerning the possible PoS realizations of the given string need one more argument and are therefore, e.g., `eval(Kind, Kindergarten, 2)` [child, kindergarden] or `eval(rot, Roggenbrot, 1)` [red, rye bread].

Once called, the program communicates with our database via interface. Our database contains not only a dictionary of all German word forms and the affix list, but also a list of search argument strings and the test corpus where the search is performed on.

Our affix lists for the German language were built upon the affix lists by [15] and [7] and expanded by us. They now contain 116 affixes.
The list of permissible consonant clusters for the German language was built upon [14] and [11] and includes 20 consonantal phonemes, 30 consonant clusters for the beginning and 72 for the end of a word.
We used a German dictionary containing about 240,000 entries. We didn't take capitalization into account.

Figure 5 shows a class diagram with the operations that can be used for interaction with the database.



```
Interface
DB

dictionaryContainsWord (String word): bool
containsAffix (String word): bool
getRule (String affix, int type): int
getAffix (int type): ResultSet
getAffixClass (int type, int class): ResultSet
```

Figure 5. Class diagram: Database interface

Interaction with the database is possible using the following commands:

- `dictionaryContainsWord()` looks up a given string in the dictionary.
- `containsAffix()` looks up a given string in the affix lists.
- `getRule()` returns which PoS class may be created with a given affix and a PoS realization type
- `getAffix()` returns a list of possible affixes for a given PoS realization type
- `getAffixClass()` returns a list of possible affixes for a given PoS realization type and a PoS class

All algorithms were implemented in PHP 5.2.4. We used MySQL 5.0.51a as a database. All scripts were executed on an Apache 2 webserver on Ubuntu 8.04, 2.2GHz, 2GB RAM.

6. EVALUATION

In this section, we describe the complete evaluation process. First, we explain what elements were chosen for evaluation purposes and how the test corpus was set up. Second, we define our measures. Third, we present the evaluation results, which are then discussed. We also discuss limitations of the results and our approach.

6.1 Test Corpus

For the evaluation, we need two corpora: on the one hand, we need a list of given lexemes that show the differences between our algorithms; on the other hand, we need a text corpus where those lexemes can be found in all their appearance forms.

The list of lexemes was built choosing lexemes based on following conditions:

- the lexeme should be a single unit of meaning
- the same string should also appear in words with a different meaning
- evaluation results for the given lexeme should not be the same for all seven algorithms.

Like [9], we have decided to handle irregular morphology like Ablaut in the lexicon. For example, the irregular verb *riechen* [smell] is stored in its two allomorphic states *riech* (present stem) and *roch* (past stem). We started with a handcrafted list with several hundred lexemes, which also appear in words with a different meaning. After the first round of evaluation, we excluded all lexemes that were evaluated with the same results by all our algorithms (either all true or all false) and only kept lexemes that caused inhomogeneous evaluation results. That led to a number of 16 lexemes, which can be found in Table 3.

We used that list on a text corpus in order to identify different words including the lexeme or its string. From a German corpus with 3 million phrases[2], we filtered out all words that contain one of the 16 strings, which was the case for 1800 different words[3]. In a next step we manually excluded proper names and foreign words, which led to a list of 1657 remaining words.

Of course, evaluation on such a corpus necessarily leads to a very high error rate. This is chosen with intent to show the bigger differences between the algorithms that are subject of interest. Both, the corpus and the list of lexemes, were stored in the database.

Table 3. Lexemes chosen for evaluation

Semantic unit	Classification	English Translation
biss	N/V	bit/bite
blau	A	blue
bläu	A	blue
find	V	find
flach	A	plane
hell	A	lucid
hör	V	hear
kling	V	sound
lampe	N	lamp
laut	N	loud
riech	V	smell
roch	V	smell
rot	A	red
röt	A	red
sah	V	see
seh	V	see

Furthermore, a training set was needed for the algorithm using supervised learning. Algorithm 6 was trained on a training set of 642 entries out of the 1657 that were annotated manually by the authors, according to the structure as described in Section 4.6. The training set was defined manually and set up in two steps: First, the semantic relation between each word and the given lexeme was evaluated manually, e.g. *[Wangenrötung, röt, 1]* or *[Schildkröten, röt, 0]*. Second, each word was annotated manually according to the word fragmentation schema (as described in Figure 4). The decision tree is induced according to [10].

6.2 Measures

Each algorithm is executed separately. The evaluation results from each algorithm are stored together with their runtime. Algorithms can hence be compared with regard to runtime and error rates in comparison to the manually defined results.

Errors can be divided into two different groups:

[2] http://corpora.uni-leipzig.de/downloads/deu_news_2008_3M-mysql.tar

[3] 'Different' is meant as a difference of at least one letter.

- Error #1: The evaluation result by the algorithm is false positive, which means evaluation is positive whereas the correct evaluation result is negative.
- Error #2: The evaluation result by the algorithm is false negative, which means evaluation is negative whereas the correct evaluation result is positive.

For query analysis, error #1 is the somehow less serious problem; in worst case, some not relevant search results are shown to the user who has to identify relevant hits anyways. Error #2 is the more problematic error as relevant search results are sorted out and can hence not be accessed by the user.

The error rate is calculated as the sum of errors of both classes.

Runtime was calculated as the arithmetic mean of ten runs for each algorithm. The measures for runtime are relative values: Algorithm 0 is the fastest; its runtime is normalized to 1. The runtime for the other algorithms is expressed in relative values, e.g. algorithm 5 is 20 times slower than algorithm 0.

6.3 Results

In this section, we present the results from our evaluation. This section shows how well each algorithm works and discusses the findings.

Each algorithm is analyzed for its error rate and for runtime. The error rate (allover, error #1, error #2) is given in absolute and relative numbers. The evaluation results are shown in Table 4.

Table 4. Evaluation results

	Error rate		**Error #1**		**Error #2**		**Runtime**
	abs.	%	abs.	%	abs.	%	
Algorithm 0	798	48,16	798	48,16	0	0	1
Algorithm 1	285	17,2	156	9,42	129	7,79	79,02
Algorithm 2	271	16,35	193	11,65	78	4,71	60,92
Algorithm 3	251	15,15	147	8,87	104	6,28	122,27
Algorithm 4	263	15,87	165	9,96	98	5,91	209,02
Algorithm 5	489	29,51	474	28,61	15	0,91	20,11
Algorithm 6	289	17,44	136	8,21	153	9,23	66,28
Number of words in the evaluation: 1657							

Out of 1657 words in our corpus, by manual evaluation, 859 were thought to be evaluated with a positive and 798 with a negative result.

908 words were evaluated with the correct result by all algorithms (except the naive algorithm 0), of which 602 were correctly evaluated positively and 306 negatively.

Correct positive evaluation was mostly accomplished for simple compounds with a maximum of two constituents (with or without word joint) or one suffix. Very few words with more than one suffix were evaluated correctly by all algorithms, which is not surprising as most of the algorithms use the same procedure for prefix handling. The number of prefixes does not seem to influence the result. Correct negative evaluation was mostly accomplished for strings that occur haphazardly in a word, which is the case for e.g. *rot* [red] in *Brot* [bread], *Protokoll* [protocol] or *Schrott* [scrap], or for *riechen* [smell] in *kriechen* [crawl] or *Griechen* [Greek]. Also most of the compounds including such strings were evaluated correctly.

However, there are several words where all algorithms failed and produced error #1 results, e.g. *rot* [red] in *neurotisch* [neurotic]; *seh* [see] in *Sehne* [tendon], *sehnen* [yearn] or compounds with the word joint *s* plus *Ehe* [marriage]; or *kling* [sound] in *Klinge* [blade]. The problem lies in the fact that for those words, the letters around the given lexeme can be matched completely with affix and dictionary entries and are hence evaluated positively.

No words evoked error #2 results in every algorithm.

6.4 Discussion

Due to our decision to only include difficult words in our evaluation, the error rates are many times higher than they would be when analyzing standard corpora.

Comparison with the naïve algorithm 0 shows that error rates can be drastically decreased with the help of intelligent evaluation algorithms. Already the phonetic evaluation (algorithm 5) can reduce the error rate for about 40 percent of algorithm 0's error rate whereas the dictionary based algorithms (algorithms 1-4, 6) show error reductions to only a third.

A detailed analysis of the results showed the following strengths and weaknesses of specific algorithms:

Algorithm 1 and 2 do not show any special dissenting results. This may easily be explained by the fact that most of the other algorithms are built upon the structure of those two algorithms.

Algorithm 3 is performing best concerning the absolute error rate because of its rules for suffix use. At the same time, those rules are not covering all cases and may sometimes lead to an exclusion of relevant matches (error #1), e.g. the suffix *ung* [-ing] may, according to our classification concerning the PoS realization type, only be combined with verbs and nouns and hence excludes *Rötung* [reddening] or the participle *erhellt* [illuminated], which actually would demand a more complex definition of state changes because *erhellt* cannot be deduced directly to *hell* [lucid], but only to the verb *erhellen* [illuminate].

Algorithm 4 was the only algorithm that produced an error #2 for the verb *klingeln* [to ring] and all its derivatives in relation to the semantic unit *klingen* [sound].

Algorithm 5 has a really low number of error #2 results, but a very high number of error #1 results because of the positive evaluation of lexemes that are surrounded by valid consonant clusters, e.g. for *rot* [red] *Karotte* [carrot], *Erotik* [erotic], or *Knäckebrot* [crispbread]. Especially the last example shows the problematic reduction to consonant clusters: *Knäckebrot* (consonant cluster: *b*) in relation to *rot* is evaluated incorrectly positively whereas *Roggenbrot* [rye bread] (consonant cluster: *nb*) oder *Brot* [bread] (consonant cluster: *b*, but without any preceding vowel) are evaluated negatively.

Algorithm 6 shows some errors due to an apparent lack of adequate examples in the training set. The problem lies yet not in the algorithm, but in the composition of the training set.

The results for runtime are thought as a rough orientation and should not be misinterpreted as disqualification criterion for the slower algorithms. On the one hand, the absolute runtime lies between 0.05 and 10.50 seconds, which remains within reasonable bounds for all algorithms. On the other hand, the performance strongly depends on the implementation of the algorithms, the used programming language and the technical equipment; e.g. algorithm 3 might perform better in a logic oriented programming language, and algorithm 4 would be likely to perform better if implemented in a programming language with better string manipulation than PHP.

Overall, the choice of the algorithm should depend on the specific problem that needs to be solved. If it is crucial to include all potentially relevant search results, one should prefer an algorithm with a low error #2 rate, which is the case for algorithm 5. In cases where the number of search stems is quite small, one could use one of the algorithms described above and assign exception lists manually. However, for larger stem sets a combination of several approaches seems to be promising.

6.5 Limitations

Of course, our grammatical approach can point out semantic relation only to a certain extent. In our approach, we do not take semantic changes caused by the combination of the lexeme with different affixes or lexemes into account. The word *sprechen* [speak] can, e.g., occur as *vorsprechen* [audition] or *nachsprechen* [repeat sth. after sb.] where the semantic relation is perfectly clear. But this is not any more the case for *entsprechen* [comply], which still could be reduced to *sprechen* when only looking at the grammatical aspect, but actually has little relation to the verb *sprechen* in means of a communicative act.

Another reason for misclassification can be rule-consistent, but incorrect compound splitting according to grammatical schemata; e.g. *neurotisch* [neurotic] has no semantic relation to *red* [rot], but can be incorrectly split up into the semantic units *neu* [new] + *rot* [red] + *isch* [ic].

Such problems could partly be solved by definition of exceptions and exception rules as suggested by [15]. Such an approach demands an extensive investigation of manual work, but would make the error rate decrease, especially for errors #1.

Another topic that has not been solved with our algorithms is the handling of polysemes: compounds that are ambiguous either because they include ambiguous components or because their semantic content depends on their pronunciation, would need some special treatment. As our algorithms only work on a lexical basis, we did not take disambiguation problems into account. This could of course be tackled when moving from a lexical to a syntactical examination level.

7. CONCLUSIONS AND OUTLOOK

In this paper, we examined the semantic relation between a given lexeme and a compound which includes that lexeme in compounding languages. Using German as an example language, we developed several algorithms, all working with a twofold input: a given lexeme and a compound including that lexeme. For testing these algorithms, we built a corpus, which only contains lexemes that cause problems to at least one of the algorithms. Hence, error rates are many times higher than they would be when analyzing standard corpora, but at the same time such testing allows a more detailed comparison of differences within the results in order to understand the strengths and weaknesses of the different algorithms.

As expected, the error rate decreases proportionally to the number and complexity of linguistic rules used by the algorithm. Whilst phonetic evaluation already reduces the error rate for about 40 per cent of algorithm 0's error rate, the dictionary based algorithms show reductions to only a third. Still, each algorithm has its specific strengths and weaknesses. The results could be improved with the definition of exceptions and exception rules.

The work presented in this paper is relevant to query analysis and term weighting approaches in information retrieval system design for the filtering of relevant search results. The results of this study ground further investigation in order to build a new compound splitting algorithm that combines the strongest aspects of the algorithms described in this paper.

8. ACKNOWLEDGEMENTS

Funding for this research was partly provided by the fFORTE WIT – Women in Technology Program of the Vienna University of Technology.

9. REFERENCES

[1] Alfonseca, E., Bilac, S., Pharies, S.: Decompounding query keywords from compounding languages, in: Proceedings of ACL-08, Columbus, 2008, pp. 253–256.

[2] Alfonseca, E., Bilac, S., Pharies, S.: German Decompounding in a Difficult Corpus, Springer, Berlin, 2008, pp. 128-139.

[3] Baccianella, S., Esuli, A., Sebastiani, F.: SENTIWORDNET 3.0: An Enhanced Lexical Resource for Sentiment Analysis and Opinion Mining. Proceedings of the 7th conference on International Language Resources and Evaluation LREC10, pp. 2200–2204, 2008.

[4] Bozsahin, C.: The Combinatory Morphemic Lexicon, Middle East Technical University, Turkey, 2002.

[5] Braschler, M., Ripplinger, B.: How Effective is Stemming and Decompounding for German Text Retrieval?, in: Information Retrieval 7, 2004, pp. 291–316.

[6] Brown, R.D.: Corpus-Driven Splitting of Compound Words, in: Proceedings of the TMI 2002, Keihanna, Japan, 2011, pp.12–21.

[7] Canoo Engineering AG: Deutsche Wörterbücher und Grammatik. Available at: http://www.canoo.net/services/WordformationRules/ueberblick/ (Feb. 2012)

[8] Carstensen, K.-U., Ebert, Ch., Ebert, C., Jekat, S., Langer, H. and Klabunde, R.: Computerlinguistik und Sprachtechnologie, Elsevier, München, 2009.

[9] Geyken, A. and Hanneforth, T.: TAGH: A Complete Morphology for German Based on Weighted Finite State Automata. In *Proceedings of the FSMNLP 2005*, Springer, Berlin, 2006, 55–66.

[10] Gupta, G.K.: Introduction to Data Mining with Case Studies, Prentice-Hall of India, New Delhi, 2006.

[11] Hess, W.: Grundlagen der Phonetik, Rheinische Friedrich Wilhelms Universität Bonn, 2001.

[12] Holz, F., Biemann, C.: Unsupervised and Knowledge-free Learning of Compound Splits and Periphrases, Springer, Berlin, 2008, pp. 117–127.

[13] Ingason, A.K., Helgadóttir, S., Loftsson, H. and Rögnvaldsson, E.: A Mixed Method Lemmatization Algorithm Using a Hierarchy of Linguistic Identities (HOLI). In *Proceedings of GoTAL 2008*, LNAI vol. 5221. Berlin: Springer, 2008, pp. 205–216.

[14] Jürgenson, I. B.: Neuronale Korrelate phonotaktischer Verarbeitung, Dissertation: Universitätsmedizin Berlin, 2009.

[15] Kellner, G.: Wege der Kommunikationsoptimierung. Anwendung von NLP im Bereich der Künstlichen Intelligenz. VDM, Saarbrücken, 2010.

[16] Kellner, G., Berendt, B.: Extracting Knowledge about Cognitive Style. The Use of Sensory Vocabulary in Forums: A Text Mining Approach, in *Proceedings of the NLPKE 2011*, IEEE Press, 2011.

[17] Macherey, K., Dai, A.M., Talbot, D., Popat, A.C. and Och, F.: Language-independent compound splitting with morphological operations, in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, 2011; pp.1395–1404.

[18] Porter, M. F.: An algorithm for suffix stripping, in: Program, Nr.3, 1980, pp. 130–137.

[19] Porter, M.F., Boulton, R., Miles, P. et al.: Snowball Project: German Stemming Algorithm. Available at: http://snowball.tartarus.org/algorithms/german/stemmer.html (Feb. 2012)

[20] Protaziuk, G., Kryszkiewicz, M., Rybinski, H., Delteil, A.: Discovering Compound and Proper Nouns, Springer, Berlin, 2007, pp. 505–515.

[21] Stymne, S.: German Compounds in Factored Statistical Machine Translation, in *Proceedings of GoTAL*, 6th International Conference on Natural Language Processing, Springer LNCS/LNAI Vol. 5221, 2008, pp. 464–475.

[22] Williams, E.: On the Notions "Lexically Related" and "Head of a Word". *Linguistic Inquiry* 12/2, 1981, pp. 245–274.

[23] Zemb, J. M.: Vergleichende Grammatik Französisch–Deutsch. Part 1: Comparaison de deux systèmes. Part 2: L'économie de la langue et le jeu de la parole. Duden, Mannheim, 1984.