Using the Levenshtein Edit Distance for Automatic Lemmatization: A Case Study for Modern Greek and English

Dimitrios P. Lyras, Kyriakos N. Sgarbas, Nikolaos D. Fakotakis
Wire Communications Lab, Electrical and Computer Engineering Department,
University of Patras, Patras, GR-26500, Greece
d.lyras@wcl.ee.upatras.gr, sgarbas@upatras.gr, fakotaki@wcl.ee.upatras.gr

Abstract

In the present work we have implemented the Edit Distance (also known as Levenshtein Distance) on a dictionary-based algorithm in order to achieve the automatic induction of the normalized form (lemma) of regular and mildly irregular words with no direct supervision. The algorithm combines two alignment models based on the string similarity and the most frequent inflexional suffixes.

In our experiments, we have also examined the language-independency (i.e. independency of the specific grammar and inflexional rules of the language) of the presented algorithm by evaluating its performance on the Modern Greek and English languages. The results were very promising as we achieved more than 95% of accuracy for the Greek language and more than 96% for the English language. This algorithm may be useful to various text mining and linguistic applications such as spell-checkers, electronic dictionaries, morphological analyzers, search engines etc.

1. Introduction

By the term "lemmatization" we refer to the act of normalizing the form of an (inflected) word into the form used as the headword in a dictionary, glossary or index [1]. This task can be used as a pre-processing step for many natural processing applications (e.g. morphological analyzers, electronic dictionaries, spell-checkers, stemmers, etc.). It may also be useful as a generic keywords generator for search engines and other data mining, clustering and classification tools.

Performing an accurate lemmatization can be a quite difficult and time-consuming task especially for morphologically complex languages with highly inflexional structure, such as the Modern Greek language.

At present, there exist various techniques for word lemmatization. Porter's stemmer algorithm [2] removes iteratively suffixes from a given word, thus mapping it to its stem. Modifications of this rule-based technique have been applied for Scandinavian [3], Romance and Germanic languages [4]. Another common approach is the use of light-stemming techniques. These techniques are based on the stripping of a small set of suffixes and/or prefixes without attempting any pattern recognition or root extraction on the input word [5]. Dictionary-based approaches have also been developed. According to these techniques, the given word is matched with a word in a proper dictionary file [6]. Some researchers prefer the use of machine learning techniques for the automatic lemmatization task. Mooney and Califf [7] address the problem by using relational learning with decision lists, Stroppa and Yvon [8] use the paradigm of analogical learning, while Dzeroski and Erjavec's [9, 10] approach is based on stem learning. Finally, corpus-based techniques are widely used as well. These techniques are based on the co-occurrence analysis of the word variants within a particular corpus to determine which variants belong to the same class and which do not [11].

In the present study we employ the Edit Distance Algorithm (also known as Levenshtein Distance [12]) on a dictionary-based algorithm, which combines a string similarity and a most frequent inflexional suffixes model. The performance of this model is evaluated over the Modern Greek and English languages. In both cases, a dictionary (containing the most common suffixes of the language in question) and a test file were used.

The structure of the paper is as follows: in Section 2 we provide further details concerning the Edit Distance Algorithm. In section 3 we provide some information about the morphological complexity of Modern Greek and English. Section 4 presents the architecture of the proposed lemmatization algorithm and a functional overview explaining how it deals with the stemming problem. Section 5 shows the experimental setup and the results achieved and in section 6 follows a conclusion discussing about future work.

2. The Edit Distance Algorithm

Searching similar sequences of data is of great importance to many applications such as the gene similarity determination, speech recognition applications, database and/or Internet search engines, handwriting

1082-3409/07 $25.00 © 2007 IEEE
DOI 10.1109/ICTAI.2007.41

IEEE computer society

recognition, spell-checkers and other biology, genomics and text processing applications [13, 14]. Therefore, algorithms that can efficiently manipulate sequences of data (in terms of time and/or space) are highly desirable, even with modest approximation guarantees.

The *Edit Distance* (also known as *Levenshtein Distance*) [12] is an easy dynamic programming algorithm which addresses the problem of sequence matching based on the notion of a primitive edit operation. By the term "primitive edit operation" we refer to the substitution of a symbol by another symbol, the deletion of a symbol and the insertion of a symbol. It is obvious that using just the three aforementioned primitive edit operations it is always possible to transform an initial string A into a target string B (provided that both strings are subject to the same alphabet).

The Levenshtein Distance of two strings A and B is the minimum number of single character insertions, deletions and substitutions required to transform A into B. The Levenshtein Distance, denoted by $d_L(A, B)$, can be easily computed using the dynamic programming scheme shown in Table 1.

Table 1. Dynamic programming scheme for the computation of the Levenshtein distance.

$$d_L[i, j] = min\begin{cases} d_L[i-1,j]+c[A_i,\varepsilon], \\ d_L[i,-1j]+c[\varepsilon,B_j], \\ d_L[i-1,j-1]+c[A_i,B_j] \end{cases}$$

for $i \geq 1$ *and* $j \geq 1$,
with $d[0, 0] = 0$ *and*
$d_L[i,-1] = d_L[-1, j] = \infty$

where:
A_i *is the* i^{th} *element of the string A*
B_j *is the* j^{th} *element of the string B*

Supposing that the strings A and B consist of n and m number of characters respectively, then the commonly used bottom-up dynamic programming algorithm for computing the Levenshtein Distance between those two strings would require the use of a (n+1)×(m+1) matrix. Table 2 presents a pseudo-code version of the Edit Distance algorithm and Table 3 shows the matrix produced when the Levenshtein Distance between the English phrases "I am old" and "I was cold" is calculated.

In the illustrative example shown in Table 3, we may observe that the first two characters of both strings match exactly, so the Levenshtein distance until that point is zero. On the other hand, when the process reaches the third character, then a mismatch occurs corresponding to a primitive edit operation (insertion), thus the edit distance between the two strings increases by one. This procedure is continued until all the characters are examined, resulting into an overall Levenshtein distance of 3 operations between the two strings (i.e. the minimal number of edit operations that are needed in order to transform the string "I am old" into the string "I was cold" is 3).

Table 2. A pseudo-code version of the standard Levenshtein distance algorithm.

```
int LevenshteinDistance(char s[1..m], char t[1..n])
// dL is a table with m+1 rows and n+1 columns
declare int dL[0..m, 0..n]

for i from 0 to m
    dL[i, 0] := i
for j from 1 to n
    dL[0, j] := j

for i from 1 to m
    for j from 1 to n
        if s[i] = t[j] then cost := 0
                       else cost := 1
        dL[i, j] := minimum(
                    dL[i-1, j] + 1,        // deletion
                    dL[i, j-1] + 1,        // insertion
                    dL[i-1, j-1] + cost // substitution
                                )
return dL[m, n]
```

Table 3. The matrix created after the calculation of the Levenshtein distance between the English phrases "I am old" and "I was cold".

		I		*a*	*m*		*o*	*l*	*d*
	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
I	*1*	***0***	*1*	*2*	*3*	*4*	*5*	*6*	*7*
	2	*1*	***0***	*1*	*2*	*3*	*4*	*5*	*6*
w	*3*	*2*	***1***	*1*	*2*	*3*	*4*	*5*	*6*
a	*4*	*3*	*2*	***1***	*2*	*3*	*4*	*5*	*6*
s	*5*	*4*	*3*	*2*	***2***	*3*	*4*	*5*	*6*
	6	*5*	*4*	*3*	*3*	***2***	*3*	*4*	*5*
c	*7*	*6*	*5*	*4*	*4*	***3***	*3*	*4*	*5*
o	*8*	*7*	*6*	*5*	*5*	*4*	***3***	*4*	*5*
l	*9*	*8*	*7*	*6*	*6*	*5*	*4*	***3***	*4*
d	*10*	*9*	*8*	*7*	*7*	*6*	*5*	*4*	***3***

3. Morphological Complexity in Modern Greek and English words

This section gives a simplified overview of the inflexion, derivation and word composition patterns for Modern Greek and English.

3.1. The Modern Greek Language

According to the Modern Greek Grammar [15], all Greek words are classified into the following categories: articles, nouns, adjectives, pronouns, verbs, participles, adverbs, prepositions, conjunctions, exclamations and grammar particles. By these 11 parts of speech of the Greek language the six: articles, nouns, adjectives,

pronouns, verbs, participles and adverbs are declinable (i.e. each word that belongs to anyone of these six parts of speech is presented in the speech with various forms), whilst the remainder parts of speech are un-declinable (i.e. the words that belong to these categories are always presented with the same form). In order to better understand the morphological complexity of the Greek language and the wealth of the possible word variants that may appear, some more detailed information is provided concerning some of the declinable parts of speech.

In Modern Greek, nouns are characterized by their gender (masculine, feminine or neuter), their number (singular and plural) and their case (nominative, genitive, accusative and vocative) [16]. Thus a noun may appear in 8 different grammatical forms. Furthermore, apart from the different inflected suffixes, an accent transposition (stress shift) may occur in some cases. The stress mark appears in the written form of words as well. Thus, there exist 89 different inflexional categories for (regular) Greek nouns (32 for the masculine nouns, 21 for the feminine and 36 for the neuter ones).

The Greek adjectives are also characterized by their gender, their number and their case. The main difference between the Greek nouns and the adjectives is that the latter are formed in the three gender types. As a result, each adjective may appear in 24 different grammatical types. In addition, adjectives have also 3 degrees of comparison (positive, comparative and superlative). In Modern Greek there are 22 different inflexional categories for the (regular) adjectives.

The morphological complexity rises even further when it comes to verbs. The Greek verbs are characterized by their voice (active and passive), their mood (indicative, subjunctive, imperative, infinitive and participle), their tense (present, present perfect, imperfect, past, past perfect, future continuous, future present and future perfect tenses), their number (singular and plural) and their person (first, second and third person). Thus, every verb has 60 different grammatical types and there exist 6 different inflexional categories for (regular) verbs. Declination of Greek verbs also involves phenomena like augmentation, which is a formative whose function is to receive stress when the antepenultimate-syllable stress law causes a left-hand stress shift outside the confines of the word (e.g. *πίνω* "I drink" → ***έ**πινα* "I was dinking") [17], reduplication (i.e. duplication of a syllable – sometimes slightly altered – in some tenses), etc. There also exist several mildly irregular verbs, meaning verbs that exhibit alterations not only to their endings but also to their stem during declination (e.g. *λύ**ν**ω* "I solve" → *έλυ**σ**α* "I solved"), and there are also some highly irregular verbs (i.e. verbs that even their stems change completely during declination) e.g. *βλέπω* "I see" → *είδα* "I saw".

Finally, in the Modern Greek language the phenomenon of *polytypia* is very prominent. Polytypia exists when the same grammatical features correspond to more than one grammatical form (e.g. the nominative singular form of the word *πόλη* "town" corresponds to two different general singular forms namely, *πόλης* and *πόλεως*).

3.2. The English Language

The English language is not as morphologically complex as the Modern Greek because of the fact that in most cases the stem of the word remains unchangeable during the declination of the word (e.g. ***play*** → ***play-****s* → ***play****ing* → ***play****ed*) and the majority of the English tenses are formed using groups of words (e.g. *I play* → *I will play, I played* → *I had played*). Nevertheless, there are several morphological phenomena in the English language (such as the simple affixation, the point-of-affixation changes, i.e. elision and gemination, the internal vowel shifting and some highly irregular cases [18]) that are worth mentioning.

The simple affixation phenomenon exists when a single morpheme (*affix*) is added to the beginning (*prefix*), end (*suffix*), middle (*infix*), or to both the beginning and the end (*circumfix*) of a root form. A typical example of the simple word suffixation in the English language is the addition of the –ing suffix, e.g. *sleep* ↔ *sleep****ing***.

In cases where the affixation involves phonological or orthographical changes at the location where the affix is added, then there exists the phenomenon of point-of-affixation changes. The most common point-of-affixation changes in the English language are the elision, where a grapheme is usually omitted (e.g. *clos****e*** → *closing*) and the gemination, where a grapheme is usually duplicated (e.g. *stir* → *stir****r****ed*).

The internal vowel shifting involves systematic changes in the vowel(s) between the inflexion and the root (e.g. *sw****ee****p* → *sw****e****pt*). These changes are often, but not always, associated with the addition of an affix.

Finally, in English (as in Modern Greek) there exist some highly irregular cases concerning the declination of some words. In such cases, even the stem changes radically. A typical example of a highly irregular verb in English is the case of the verb "go" (i.e. *go* → *went*).

4. The Lemmatization Algorithm

In order to address the problem of automatic word lemmatization, we have implemented an algorithm based on the Levenshtein distance.

It is important to clarify that in the present study we have mainly focused on the grammar affixation phenomena (i.e. cases where a suffix is added to a word during its declination) rather than on semantic affixations (i.e. cases where a suffix is added to a word in order to change its initial meaning and/or its part of speech). For

instance, our algorithm would return the word "*entertain*" as the correct headword of the word "*entertained*", but it would consider mistaken the answer "*entertain*" if the input word was "*entertainment*", as the latter is a noun, it has a different meaning than "*entertain*" and usually constitutes a discrete entry (lemma) in an English dictionary.

The main idea is to filter the user's input word (source word hereafter) through a list containing all the lemmas of the language (target words hereafter). The target words are stored in the form used as headwords in a dictionary, glossary or index. In order to achieve this goal, we use two files (one for the Modern Greek language and one for the English language) containing over 30,000 lemmas each.

For each one of the target words, the similarity distance between the source and the target word is calculated and stored. When this process is completed, the algorithm returns a set of target words having the minimum edit distance from the source word. The algorithm provides the option to select the value of the approximation that the system considers as desired similarity distance (e.g. if the user enters zero as the desired approximation, then only the target words with the minimum edit distance will be returned, whereas if he/she enters e.g. 2 as the desired approximation, then the returned set will contain all the target words having a distance <=(minimum + 2) from the source word. This is very useful in cases where the source word is mildly irregular (i.e. in cases where changes happen even to the stem of the word during its declination, like for instance in ***φύγ***-*αμε* "we left" → ***φεύγ***-*ω* "I leave"), as the correct headword may have a greater Levenshtein distance than other target words.

In order to improve the performance of the lemmatization algorithm, we focused on the inflectional suffixes removal for any given inflectional word. Taking under consideration the Modern Greek and the English grammar, we have created two lists containing the most common suffixes used in each language. For Modern Greek we have also created an additional database containing the stems of the target words. Thus, the user has the ability to define the state of the input word (i.e. stemmed or normal) and the database that will be used for the mapping of the source word (i.e. stemmed database or normal database).

If the user chooses the stemming of the input word, then initially the algorithm retrieves from the appropriate suffix list all the suffixes that are possible endings of the source word. Afterwards, the first possible suffix is subtracted from the input word and a calculation between the modified entry and all the target words is performed. The target words with the minimum Levenshtein distance are stored in a file and the above procedure is repeated until all possible endings of the source word are subtracted from it. When this happens, then the algorithm compares the edit distance of all the stored target words and returns the n-best ones with the overall minimum edit distance (according to the approximation preferences.).

It is worth mentioning that in order to cope with accent transposition phenomena, stress marks are ignored both for the source word and for the target words.

Table 4 shows an example of a source word, its possible endings and part of the target words that are contained in the databases, while Figure 1 presents the logic flow diagram of the lemmatization algorithm.

Table 4. An illustrative example of a source word, its possible endings and part of the target words.

Source Word	*Source Word Unstressed*	*Possible Endings*	*Normal Database*	*Stemmed Database*
αρκούδες (bears)	*αρκουδες*	*-ες*	……	……
		-ουδες	*αρκετος*	*αρκετ*
			αρκουδα	***αρκουδ***
			αρκουδακι	*αρκουδακ*
			αρκουδι	***αρκουδ***
			αρκουδιαζω	*αρκουδιαζ*
			……	……
			αρκουδος	***αρκουδ***
			……	……
			αρκω	***αρκ***
			……	……

The problem appears when some of the suffixes in the list affect the source words in a wrong way by removing a wrong part of the input word as a suffix. In order to make the problem clearer, we will use the example source word presented earlier. As it is shown in Table 4, for the input word "*αρκουδες*" (bears) the algorithm retrieves from the Greek suffix list two possible endings, namely *–ες* and *–ουδες,* but only the first one is correct.

If the first possible suffix is subtracted from the source word then the target words with the minimum edit distance are the following: ***αρκούδ****α* (bear), ***αρκούδι*** (bear-cub) and ***αρκούδ****ος* (male bear), as all three words have the same stem (αρκουδ). On the other hand, when the second possible suffix (*–ουδες*) is removed from the source word, then the target word with the minimum edit distance is ***αρκ****ώ* (to be enough).

Unfortunately, we are not able to apply general rules (e.g. always remove the suffix *–ες*) for cases as the one just described, since there other sets of words where these general rules would be incorrect (e.g. the Greek word *αλεπούδες* (foxes) has the same possible suffixes as *αρκούδες*, namely *–ες* and *–ουδες*, but in this case the correct suffix to be removed is *–ουδες* as the stem of the word is *αλεπ*).

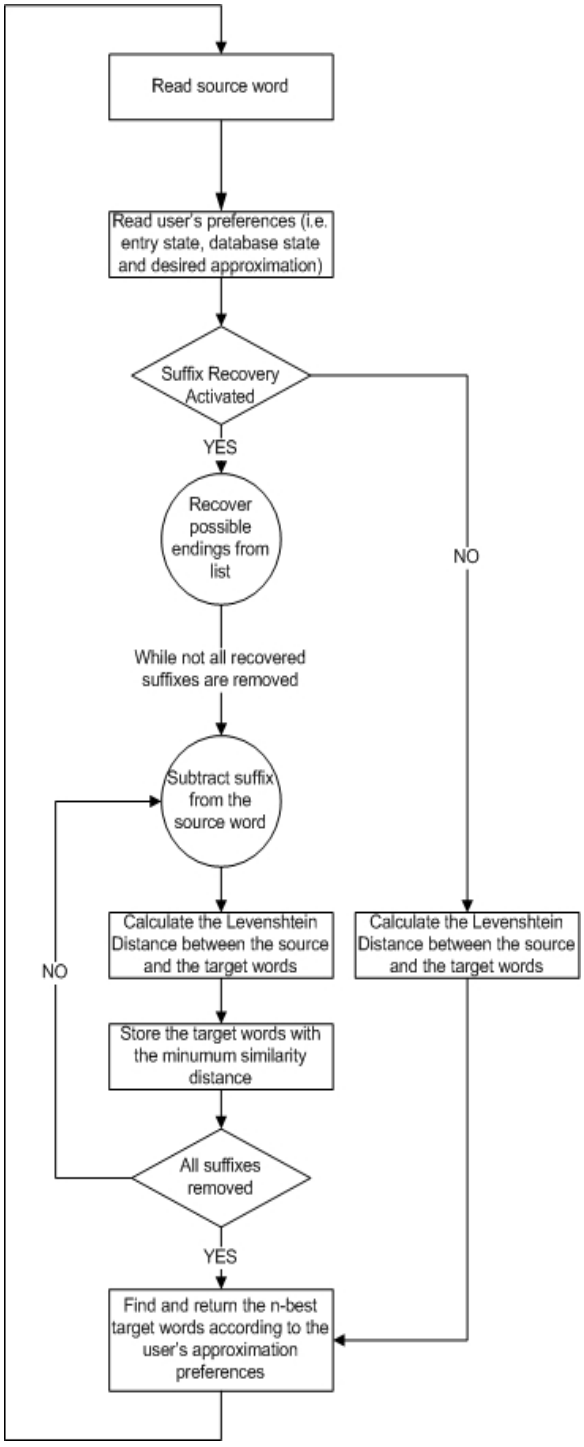

Figure 1. Logic Flow Diagram of the Lemmatization algorithm.

For the time being, the algorithm addresses the aforementioned problem by considering as correct all the target words that have a similarity distance equal to the minimum one. Thus, for the input word *αρκουδες* and with zero approximation, the returned target words would be: *αρκούδα*, *αρκούδι*, *αρκούδος* and *αρκώ*. Other approaches (e.g. create lists of the words that are wrongly affected by such rules or select the target words with the highest frequency of appearance) have also been taken under consideration and are set out at section 6.

5. Experimental Setup and Results

In order to evaluate the performance of our automatic lemmatization algorithm and test its independence from a particular language (i.e. independence of the specific grammar and inflectional rules of the examined language), experiments were performed for Modern Greek and English. All the experiments described below were conducted using a zero desired approximation, since a positive approximation value would present a better accuracy score within a larger set of returned results.

5.1. Performance Test for Modern Greek

For the experiments conducted on the Modern Greek language, the following resources were used: a) a database containing 85,203 Greek words and phrases stored in a headword form (normal database hereafter), b) a database containing the same words in their stemmed form (stemmed database hereafter) which was used in order to better address the problem of the inflectional suffix removal as described in section 4, c) a database containing 251 Greek suffixes, and d) a test file containing 1470 Greek words in various spoken forms and their correspondent headword form. Each one of these words was used as a source word in our system and the returned results were compared to the correct headword of the source word.

Obviously, the following four kinds of experimental combinations were possible: a) calculation of the Levenshtein distance between the un-stemmed form of the source word and the target words contained in the normal database (Case A), b) calculation of the Levenshtein distance between the un-stemmed form of the source word and the target words contained in the stemmed database (Case B), c) attempt the stemming of the source word and use the target words contained in the normal database for the calculation of the Levenshtein distance (Case C), and d) attempt the stemming of the source word and use the target words contained in the stemmed database for the calculation of the Levenshtein distance (Case D). In order to find out which one of the aforementioned combinations addresses the problem of the automatic lemmatization more efficiently, all of them were examined. The results of the accuracy evaluation are given in Table 5.

Table 5. Accuracy results for the experiments conducted on the Modern Greek language.

(Case A)	*Norm.*	*Stem.*	*(Case B)*	*Norm.*	*Stem.*
Source Word State	✓		*Source Word State*	✓	
Database State	✓		*Database State*		✓
Accuracy	*70.34% (1034/1470)*		*Accuracy*	*63.40% (932/1470)*	

(Case C)	*Norm.*	*Stem.*	*(Case D)*	*Norm.*	*Stem.*
Source Word State		✓	*Source Word State*		✓
Database State	✓		*Database State*		✓
Accuracy	*62.29% (916/1470)*		*Accuracy*	***95.03% (1397/1470)***	

As expected, the higher accuracy was achieved when the calculation of the Levenshtein distance was performed between the stemmed form of the source word and the target words contained in the stemmed database (Case D). It is also worth mentioning that although someone would expect the accuracy in cases B and C to be higher than in case A, this does not happen. This can be easily explained if we consider the morphological complexity of Modern Greek and the way the Levenshtein distance works. More specifically, in situations like the ones described in case A, the suffix of the un-stemmed source word may have common characters with the suffix of the correct un-stemmed target word, thus resulting in a smaller Levenshtein distance (e.g. the word *παπούτσ-ια* "shoes" has a Levenshtein distance of 1 when compared to the word *παπούτσ-ι* "shoe", whereas it has a Levenshtein distance of 2 when compared to the stemmed word *παπούτσ*).

For a given source string more than one target words may present the same Levenshtein distance (and thus all of them may be returned as candidate answers by our algorithm). Out of the 1397 correctly predicted cases, when the algorithm was tested in case D, 624 times it returned just the correct answer, 328 times 2 target words were returned (among which the correct answer was included) and only 70 times the returned set had more than 5 words. Figure 2 depicts the percentage of the cases where the algorithm returned 1, 2, 3, 4, 5 and more than 5 results (among which the correct answer was always included) when tested in case D (best-case scenario).

Although in many cases more than one target words may have the same Levenshtein distance as the correct headword (see the example presented in section 5), as Figure 2 shows, in the majority of the cases less than 3 target words were returned as candidates. It is also important to mention that when more than one target words were returned, their vast majority had a very close semantic similarity with the correct headword, e.g. for the source word *αρκούδες (bears)* the returned results would be *αρκούδα (bear)*, *αρκούδι (bear-cup)*, *αρκούδος (male bear)* and *αρκώ (to be enough)*. It is obvious that the first three returned results are semantically relevant to the correct headword *(αρκούδα)* and only the last one is completely irrelevant.

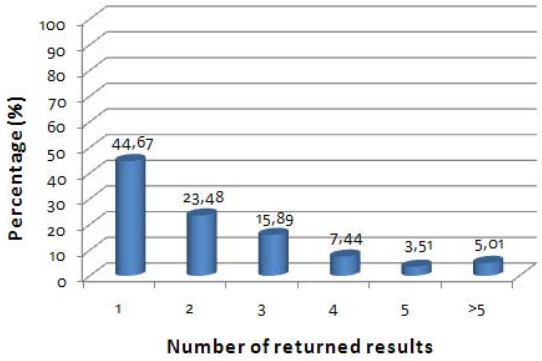

Figure 2. Percentage of cases where the algorithm returned 1, 2, 3, 4, 5, or more than 5 results, when evaluated for the Case D of the Modern Greek language experiments.

5.2. Performance Test for English

The resources used for the experiments on the English language were: a) a database containing 36,647 English words stored in a headword form (normal database hereafter), b) a database containing 26 English suffixes, and c) a test file containing 1470 English words in various spoken forms and their correspondent headword form.

The possible experimental combinations this time were just two: a) calculation of the Levenshtein distance between the un-stemmed form of the source word and the target words contained in the normal database (Case A) and b) attempt the stemming of the source word and use the target words contained in the normal database for the calculation of the Levenshtein distance (Case B). Table 6 shows the performance of the algorithm for these two cases.

Table 6. Accuracy results for the experiments conducted on the English language.

(Case A)	*Norm.*	*Stem.*	*(Case B)*	*Norm.*	*Stem.*
Source Word State	✓		*Source Word State*		✓
Database State	✓		*Database State*	✓	
Accuracy	*91.02% (1338/1470)*		*Accuracy*	***96.46% (1418/1470)***	

Two very important remarks may come up from the results presented at Table 6. The first one is that again the performance of the algorithm was significantly improved when the source words were stemmed (Case B). The second remark is that the overall accuracy is higher for English than for Modern Greek. This is mainly due to the fact that the English language is not as morphologically complex as the Modern Greek and in the majority of the cases the morphological phenomena that appear in English do not provoke radical changes to the stem of the word.

Again, we measured the performance of the algorithm according to the number of returned results, this time for the English language. In case B (best-case scenario), the percentage of the cases where 1, 2, 3, 4, 5 and more than 5 results were returned (always including the correct answer) is graphically represented at Figure 3.

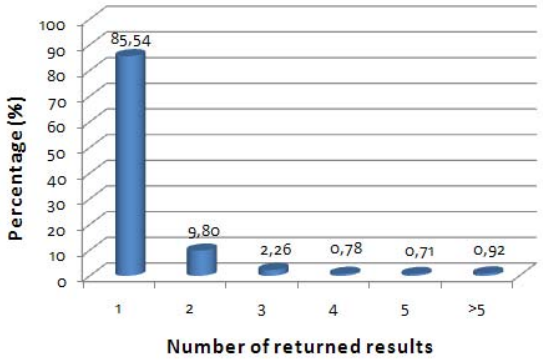

Figure 3. Percentage of cases where the algorithm returned 1, 2, 3, 4, 5, or more than 5 results, when evaluated for the Case B of the English language experiments.

As Figure 3 shows, in more than 85% of the cases, the algorithm returned just the correct target word, while in less than 1% of the cases there were more than 5 target words returned. By comparing Figures 2 and 3, we realize that the algorithm performs better for English than for Modern Greek (i.e. it is able to track the correct headword more easily for English).

6. Conclusions and Perspective

In this paper we demonstrated a new language-independent lemmatization algorithm based on the Levenshtein distance and we evaluated its performance both on Modern Greek and English languages. By the term "language-independent", we mean that the algorithm can perform sufficiently well for a variety of languages regardless of the specific grammar and inflectional rules that apply to them.

The proposed algorithm may be useful to various text mining and natural language processing applications such as spell-checkers, morphological processors, electronic dictionaries, web search engines, stemmers, identifiers of lexical features for automatic document classification etc.

The only resource required for the algorithm to function is a database containing all the words of the examined language in a headword form (lemmas). Optionally, one could also use a database containing all the suffixes of the language in question, thus improving the performance of the proposed system.

In the preliminary evaluation, the presented lemmatizer was tested against two test files (one for the Modern Greek and one for the English language) containing 1470 words each. The experimental results suggest that the proposed algorithm performs exceptionally well for the examined languages, reaching an accuracy of 95% for Modern Greek and 96% for English. In addition, the experiments confirmed that its performance depends on the morphological complexity of the particular language and that it is significantly improved when the given word undergoes an inflectional suffix removal process. Finally, another important conclusion that is derived from this study is that since more than one lexical entries may present the same Levenshtein distance as the correct headword, the number of returned results highly depends on the morphological complexity of the specific language.

A comparison of the results of the proposed system to the ones achieved by other approaches cannot be performed directly due to the different resources and test files used. Nevertheless, we may observe that the presented algorithm outperforms other language-independent techniques (e.g. use of machine learning algorithms [7, 10]) but performs slightly worse when compared to systems that use grammatical information and other multimodal alignment models (e.g. relative corpus frequency) of the examined languages, as is done in [19].

In order to improve the performance of our algorithm, various approaches have been taken under consideration. We believe that the use of grammatical information (i.e. separate rules concerning the declination of verbs, nouns, adjectives etc.) of the examined language and the ranking of the returned results according to their frequency of appearance would increase the accuracy of the lemmatizer. Nevertheless, approaches like the ones just described, would result into making the overall system more language-dependable. Therefore our primary goal for future work would be the integration of other language-independent algorithms in our system (e.g. the dice coefficient algorithm [20], machine learning techniques etc.) and the evaluation of their accuracy according to the lemmatization task when applied separately and combined with the Levenshtein distance algorithm.

7. References

[1] C. & G. Merriam Co., *Webster's Revised Unabridged Dictionary*. (1913). Retrieved May 23 2007 from http://www.thefreedictionary.com/lemmatise

[2] Porter M., *An algorithm for suffix stripping*, Program, 14(3), 1980, pp. 130-137.

[3] Dalianis, H. and Jongejan B., "Hand-crafted versus machine-learned inflexional rules: The SiteSeeker stemmer and CST's lemmatiser", LREC 2006, Genoa, Italy, 2006.

[4] "Tartarus", http://snowball.tartarus.org

[5] Al-Sughaiyer I. and Al-Kharashi I.: "Arabic Morphological Analysis Techniques: A Comprehensive Survey", *Journal of the American Society for Information Science and Technology*, Vol 55, Issue 3, 2004, pp. 189 - 213.

[6] Carlberger J., Dalianis H., Hassel M. and Knutsson O., "Improving Precision in Information Retrieval for Swedish using Stemming", In Proc. Of NODALIDA '01, Uppsala, Sweden, 2001.

[7] R.J. Mooney and M.E. Califf, "Induction of first-order decision lists: Results on learning the past tense of English verbs", *In Proc. of the 5th International Workshop on Inductive Logic Programming*, 1995, pp. 145–146.

[8] Stroppa, Nicolas and François Yvon, "An analogical learner for morphological analysis", *In Proc. of the 9th Conf. on Computational Natural Language Learning (CoNLL-2005)*, 2005, pp. 120–127.

[9] Saso Dzeroski and Tomaz Erjavec, "Learning to lemmatise Slovene words", *Learning language in logic*, Springer, pp. 69–88, 2001.

[10] Erjavec Tomaz and Saso Dzeroski, "Machine learning of morphosyntactic structure: Lemmatizing unknown Slovene words". *Applied Artificial Intelligence*, 18, 2004, pp. 17–41.

[11] Xu J. and Croft W. B., "Corpus-based stemming using cooccurrence of word variants". ACM Transactions on Information Systems, 16 (1), 1998, pp. 61-81.

[12] V. I. Levenshtein. "Binary codes capable of correcting deletions, insertions, and reversals". *Soviet Physics Dokl.*, 10, February 1966, pp. 707–710.

[13] D. Gusfield. *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, 1997.

[14] P. Pevzner. *Computational Molecular Biology*. Elsevier Science Ltd., 2003.

[15] Τριανταφυλλίδης Μανώλης, *"Νεοελληνική Γραμματική"*, Οργανισμός Εκδόσεως Διδακτικών Βιβλίων (ΟΕΔΒ), 1977.

[16] Sgarbas K., "Techniques for Automatic Morphosyntactic Analysis of Modern Greek Language", PhD Dissertation No.55, University of Patras, Dept. of Electrical and Computer Engineering, June 1997 (in Greek).

[17] Ralli Angela, "Morphology in Greek linguistics", *Journal of Greek Linguistics,* John Benjamins Publishing Company, 4 (2003), pp. 77–129.

[18] Richard Wicentowski, "Modeling and Learning Multilingual Inflectional Morphology in a Minimally Supervised Framework", Johns Hopkins University Ph.D. thesis, October 2002.

[19] David Yarowsky and Richard Wicentowski, "Minimally Supervised Morphological Analysis by Multimodal Alignment", *In Proc.of the ACL-2000*, Hong Kong, 2000, pp. 207 - 216.

[20] Brew, C. and D. McKelvie, "Word-pair extraction for lexicography", *In Proc.of the 2nd International Conference on New Methods in Language Processing*, eds. K. Oflazer and H. Somers, Ankara: Bilkent University, 1996, pp. 45-55.