Natural Language Engineering (2023), **29**, pp. 1305–1340
doi:10.1017/S1351324922000018

ARTICLE

SoundexGR: An algorithm for phonetic matching for the Greek language

Antrei Kavros[1] and Yannis Tzitzikas[1,2,*]

[1]Computer Science Department, University of Crete, Heraklion, Crete, Greece and [2]Institute of Computer Science, Foundation for Research and Technology – Hellas (FORTH), Heraklion, Crete, Greece
[*]Corresponding author: E-mail tzitzik@ics.forth.gr

(Received 5 September 2020; revised 3 January 2022; accepted 3 January 2022; first published online 4 February 2022)

Abstract

Text usually suffers from typos which can negatively affect various Information Retrieval and Natural Language Processing tasks. Although there is a wide variety of choices for tackling this issue in the English language, this is not the case for other languages. For the Greek language, most of the existing phonetic algorithms provide rather insufficient support. For this reason, in this paper, we introduce an algorithm for phonetic matching designed for the Greek language: we start from the original Soundex and we redesign and extend it for accommodating the Greek language's phonetic rules, ending up to a family of algorithms, that we call $\texttt{Soundex}_{GR}$. Then, we report various experimental results showcasing how the algorithm behaves in different scenarios, and we provide comparative results for various parameters of the algorithm for revealing the trade-off between precision and recall in datasets with different kinds of errors. We also provide comparative results with matching using stemming, full phonemic transcription, and edit distance, that demonstrate that $\texttt{Soundex}_{GR}$ performs better (indicatively, it achieves F-Score over 95% in collections of similar-sounded words). The simplicity, efficiency, and effectiveness of the proposed algorithm make it applicable and adaptable to a wide range of tasks.

Keywords: Phonetic algorithms; Greek language; Phonetic matching; Phonemic matching

1. Introduction

Misspelled and mispronounced words can negatively affect various tasks in Information Retrieval (IR), and Natural Language Processing (NLP) tasks such as indexing, retrieval, autocompletion (Fafalios *et al.* 2012), entity recognition (Yadav and Bethard 2018), question answering (Dimitrakis *et al.* 2019), structured data integration (Mountantonakis and Tzitzikas 2019), and phonetic interfaces in general (Kaur *et al.* 2020). Moreover, the existing approaches for producing word embeddings (like Word2Vec Mikolov *et al.* (2013), Glove Pennington *et al.* (2014), and BERT Devlin *et al.* (2018)) have limited applicability to malformed texts, which contain a non-negligible amount of out-of-vocabulary words (Piktus *et al.* 2019), meaning that they cannot provide embeddings for words that have not been observed at training time.

To tackle such cases, *stemming* and *edit*-related distances (e.g., the Levenstein distance Levenshtein (1966)) are usually employed (e.g., Medhat *et al.* (2015)). However, these methods are not always sufficient: we cannot apply stemming to person and location names, while the edit distance between a word and a misspelled one (that has more than one misspellings) can be too big (e.g., the edit distance between "Schumacher" and "Soumaher" is 4), thus limiting the value of *edit distance*-based matching. Another family of algorithms to deal with this issue is the family of *phonetic matching* algorithms. Indeed, *phonetic codes* have been used in various contexts, for example, for indexing and retrieving names from a large dataset (Koneru *et al.* 2016), for SMS retrieval (Pinto *et al.* 2012), for link discovery (Ahmed *et al.* 2019), for duplicate record detection (Elmagarmid *et al.* 2006), and for preserving privacy (Karakasidis and Verykios 2009) and others.

The first implementation of phonetic algorithms dates back to 1918, with the Soundex algorithm (Russell 1918; Russell 1922), which attempts to encode words based on how they sound. Although there is a plethora of proposed solutions for tackling this issue in the English language (Soundex, Metaphone, Double Metaphone, Metaphone 3, NYSIIS and others), this is not the case for the Greek language. In this paper, we propose and evaluate an algorithm that falls to that family and aims at dealing with such issues for the Greek language. Such an algorithm should be able to tackle a wider variety of errors with high accuracy. For example, for the word έτοιμος (which is spelled correctly and sounds [étimos]), it should be able to retrieve (match) misspelled variations of the same word and word sense, like έτιμος ([étimos]), έτιμς ([étims]), αίτημος ([étimos]), or similar terms of a different sense like έντιμος ([éntimos]). Hereafter, we shall use [ ] to enclose both phonetic and phonemic word transcriptions.

Our approach for designing such an algorithm is to adapt the basic idea of Soundex to the characteristics of the Greek language, for having a baseline method, and then to widen its rules, like most modern (post-Soundex) phonetic algorithms have done, for accommodating the Greek language's phonetic rules. To this end, we introduce a family of algorithms that we call $\texttt{Soundex}_{GR}$. With $\texttt{Soundex}_{GR}$, we achieve assigning the same code to set of words that should match, like the set of words { μήνυμα, μύνημα, μίνιμα, μοίνειμα}, the set { εύδοξος, εβδοξος} and the set { ευάερο, αιβάερρο, αιββάαιρο}.

Then, we report comparative experimental results that show which variation/configuration of the algorithm behaves better in the evaluation over datasets with various kinds of errors. Specifically, the original Soundex algorithm, modified for corresponding to the Greek Alphabet, achieves an average F-Score equal to 0.64 across different type of errors (letter addition, deletion, or substitution). The enhanced version that takes into consideration also the Greek phonetic rules achieves an average F-Score of 0.66. The variation that uses both of the previous versions to find a match achieves an average F-Score of 0.70, while in a dataset that contains similar sounded words it reaches F-Score equal to 0.91, while $\texttt{Soundex}_{GR}$ achieves F-Score equal to 0.97. In addition, we report comparative experimental results with stemming and full phonetic transcription that show that the proposed algorithm performs better. We also evaluate how the code length affects the F-Score in datasets of different sizes, types of errors, and word lengths, and we measure efficiency by applying it over a Greek dictionary. Overall, the effectiveness, the simplicity, and the efficiency of the proposed family of algorithms makes it applicable to a wide range of tasks.

Although there are works about the phonetic (and phonemic) transcription of Greek words (e.g., Themistocleous (2011)), to the best of our knowledge, there is no work on using and evaluating such codes for matching Greek text.

The rest of this paper is organized as follows. Section 2 describes the background and discusses related work. Section 3 describes the proposed family of algorithms and provides various application examples for revealing the differences of these variations. Section 4 focuses on evaluation, presents extensive comparative results (for various datasets, codes sizes, and matching methods including stemming, full phonetic transcription, and edit distance), and discusses applications. Finally, Section 5 concludes the paper and identifies issues that are worth further work and research.

2. Background and related work

A wide variety of phonetic algorithms exist, many if not all, are descendants of the Soundex algorithm (described in detail in Section 2.1), like Philips (1990); Hood (2002). These algorithms aim at retrieving misspelled words and improving IR, by generating a coding of the query based on phonetic pronunciation rules. They are in use mainly in Database Systems to aid in the retrieval process, as well as in various IR tasks, such as indexing, query autocompletion, and retrieval. They are also useful in NLP tasks like Named Entity Recognition and Linking and word sense disambiguation in general. Unfortunately, most of them provide at best minimal or no support at all for the Greek language.

The works that concern the processing of the Greek language in general are not excessive (see Papantoniou and Tzitzikas (2020) for a recent survey), in comparison to the English language. However, there are quite a few works on the phonetics of the Greek language which are described in brief below.

The 1972 book Newton (1972) studies Greek phonology in general, while Epitropakis *et al.* (1993) presents an algorithm for the generation of intonation (F0 contours) for the Greek Text-To-Speech system. Fourakis *et al.* (1999) analyzes the acoustic characteristics of the Greek vowels (duration, fundamental frequency, amplitude, and others). Along the same line, Sfakianaki (2002) analyzes the acoustic characteristics of Greek vowels produced by adults and children, while Trudgill (2009) focuses on the Greek dialect vowel systems. Arvaniti (2007) describes the 2007 state of the art in greek phonetics.

IPAGreek (Themistocleous 2011) is an implementation (available at Themistocleous (2017)) of Standard Modern Greek and Cypriot Greek "phonological grammar." The application enables users to transcribe text written in Greek orthography into the International Phonetics Alphabet (IPA).

Karanikolas (2019) proposes an automatic machine learning approach that learns rules of how to transcribe Greek words into the International Phonetics Association's (IPA's) phonetic alphabet; however, the suggested method has not been implemented, nor evaluated.

Finally, Themistocleous (2019) describes classification approaches based on deep neural networks for distinguishing two Greek dialects, namely Athenian Greek, the prototypical form of Standard Modern Greek and Cypriot Greek. That work is based on the acoustic features of spoken language.

Most of the above works focus on acoustic aspects of the language, fewer on the management of Greek text, and in particular on the problem of retrieval and matching. One algorithm that could be used for the Greek language, and for the tasks that we mentioned, that is, for matching over Greek text, is the Beider-Morse Beider (2008), by changing Greek letters to their equivalent English letters, without taking into consideration Greek phonetic rules, but based on how they would sound in the American dialect. Another approach would be to take a phonemic transcription method, like the one described in Themistocleous (2017), and truncate and/or modify it (i.e., group different letters to the same code as a means to assist matching), for being suitable for approximate matching.

In this paper, we attempt to fill this gap in the literature, that is, propose a general purpose algorithm for phonetic matching for Greek text, and we evaluate its potential for matching in various datasets and under different configurations.

2.1 The original Soundex algorithm

As mentioned in the introduction, our approach for designing an algorithm for phonetic matching for the Greek language is to adapt the basic idea of Soundex to the characteristics of the Greek language, for having a baseline method, and then to widen its rules for accommodating the Greek language's phonetic rules.

Originating back in 1918, developed by Robert C. Russell and Margaret King Odell, Soundex algorithm had a simple set of rules. It generates a code by ignoring vowels and the letter *h* if not at the start of the word, and encoding consonants based on how they sound, generating a code of just four characters length. Specifically, the steps of the original Soundex algorithm are

(i) keep the first letter unencoded,

(ii) remove all occurrences of a, e, h, i, o, u, w, y, except when they appear as the first letter of the word,

Table 1. Consonants Replacement in the Soundex

$b, f, p, \upsilon \rightarrow 1$
$c, g, j, k, q, s, x, z \rightarrow 2$
$d, t \rightarrow 3$
$l \rightarrow 4$
$m, n \rightarrow 5$
$r \rightarrow 6.$

(iii) replace consonants after the first letter as shown in Table 1,
(iv) remove adjacent duplicate digits,
(v) produce a code of the form Letter Digit Digit Digit by ignoring digits after the third one (if needed), or by appending zeros (if needed).

For example, the name SMITH will be encoded to S530 as well as the names SCHMIDT and SMYTH, while both ROBERT and RUPERT yield R163. However, imprecise results are possible, for example, BLACK and BAILS yield the code B420.

Christian (1998) described the problems to the original Soundex, ignoring different spellings of letters in different contexts and letter combinations. Other issues include the ignoring of vowels if not at the start of the word and the short generated code. All these issues greatly harm Soundex precision levels.

The first usage of the algorithm was to retrieve people names from a large dataset, while today Soundex algorithm or its descendants can be found in various systems, for example, for SMS retrieval (Pinto *et al.* 2012), for indexing names (Raghavan and Allan 2004), for link discovery (Ahmed *et al.* 2019), for duplicate record detection (Elmagarmid *et al.* 2006), for record linkage (da Silva *et al.* 2020), etc.

Moreover, it has been adapted for various languages, including the Thai language (Karoonboonyanan *et al.* 1997), the Arabic language (Yahia *et al.* 2006; Shedeed and Abdel 2011; Ousidhoum and Bensaou 2013), the Vietnamese language (Nguyen *et al.* 2008), the Chinese language (Li and Peng 2011), the Indian language (Shah 2014; Gautam *et al.* 2019), the Assamese language (Baruah and Mahanta 2015), the Spanish language (del Pilar Angeles *et al.* 2015), and others.

2.2 Other related algorithms

Several algorithms after Soundex, sprawled from the core idea of it, group letters by their pronunciation, aiming at improving the original algorithm. Some of the most renowned ones are

Metaphone (Philips 1990): Applies a transformation to the original word, before the word is encoded through letter pronunciation buckets and a vast set of phonetic rules. Subsequently, various improvements were made to it: Philips (2000) creates a primary and a secondary encoding for a given word and applies rules based on the origin language of the input word, while Philips (2013) added configurable rules to the algorithm, as well as it further improved foreign word retrieval.

Caverphone (Hood 2002): Applies transformations to the word that may be larger than 2-gram at a time to produce an encoding. It was originally created based on accents in a specific area of New Zealand.

BMPM (Beider 2008): Before a set of phonetic rules are applied to the word, there is an identification process of the origin of the word, and then the corresponding language rules are applied.

MRA (Match Rating Approach) developed by Western Airlines in 1977 had a simple set of phonetic rules, providing through a set of comparison rules to go along the encoding.

Other phonetic algorithms produce more than one encoding to the word in order to enhance Soundex retrieval. In general, these algorithms aimed to cope with the shortcomings of the original Soundex that were described in Section 2.1 and improved it, as Koneru *et al.* (2016) suggest, in terms of precision which is the main shortcoming of the Soundex algorithm.

3. The algorithm $\texttt{Soundex}_{GR}$ (and variations)

This section is organized as follows: at first, in Section 3.1, we describe in brief the requirements. Then in Section 3.2, we describe the basic idea of the new algorithm that we call $\texttt{Soundex}_{GR}$, while in Section 3.3 we detail the exact steps of this algorithm. For reasons of comparative evaluation, in Section 3.4, we define a variation that we call $\texttt{Soundex}_{GR}^{naive}$ that shares the same principles of the original Soundex algorithm but without any word preprocessing before the encoding of the word. Finally, in Section 3.5, we introduce another variation for phonetic matching ($\texttt{Soundex}_{GR}^{comp}$) that uses both $\texttt{Soundex}_{GR}$ and $\texttt{Soundex}_{GR}^{naive}$.

3.1 Requirements for the Greek language.

The basic idea of the original Soundex algorithm can be easily translated to a Greek version. Indeed, a simple version would be to adopt the exact same rules as Soundex, as described in section 2.1, with Greek consonants. However, we wanted to tackle the shortcomings of the original Soundex (described in Section 2.1), hence to consider letter contexts, letter combinations, and generally grammar rules specific to Greek. Moreover, while the original Soundex was implemented for use mainly on names, we would like an algorithm for regular words as well. This means that we would like to achieve high precision for regular (frequent) words (to avoid having a lot of words having the same code), while for names we would like to achieve high recall (i.e., low percentage of false-positives), since they occur more rarely.

For example, we would like an algorithm that would correctly group θάλασα with θάλασσα (both sound [θálasa]), μήνυμα with μύνημα (both sound [mínima]), αίτημα with έτιμα or έτοιμα (all sound [étima]), and εύκολα with έφκολα (both sound [efkola]). The algorithm should retrieve all such cases with minimal noise and as high recall as possible.

3.2 The basic idea of $\texttt{Soundex}_{GR}$

Here, we describe our algorithm that we call $\texttt{Soundex}_{GR}$. As in the original Soundex, we keep an encoding length of just four characters. As we shall see in the experiments reported in Section 4.5, if we increase the length from 4 to 5 we get a higher precision by 5–10% percent; however, the recall is decreased 10–15%. However, in larger datasets, a bigger length can be more appropriate (detailed experimental results are given in Section 4.10).

As discussed in Section 2.1, Soundex has a precision issue, which originates from the combination of short code of just four characters and not taking in to account any lexical context. In order to improve the precision levels of the Soundex algorithm, we have to focus on these. On the contrary to Soundex, in $\texttt{Soundex}_{GR}$, we take into account a more rich set of rules, corresponding to the phonetic rules of Greek language. Below we describe the key points and subsequently we describe the exact steps.

Before a word is encoded, we preprocess it and generate a different word form. The preprocessing operations include identification of the cases when a vowel sounds as a consonant in Greek,

Table 2. Phonetic rules

consonant bigrams replacements	
2-gram	1-gram
μπ	b
ντ	d
γκ,γγ	g
τσ,τζ	c
πσ,πς	ψ
κς,κσ	ξ
vowel bigrams replacement	
2-gram	1-gram
αι	ε
οι,ει	ι
ου	ο

grouping of pairs of vowels based on how they sound, intonation removal, and dismantling of digrams to single letters. When this procedure finishes, the word is encoded.

For example, μπαίνω that sounds [béno], will be transformed to bενο and finally it will be encoded to b*7$, while the name Γιάννης (that sounds [jánis]) will be transformed to Γιανι and then it will be encoded to γ@97 (more examples will be given later on).

Another difference is that Soundex ignores vowels; however, `Soundex`$_{GR}$ does not ignore vowels, instead it groups them into three categories based on how they sound, in particular to α, ο, η, in order to improve the precision of the algorithm.

The last letter of the word is ignored if it is a consonant, specifically if it is a ς or ν, as it does not add much value to the word.

3.3 *The exact steps of* `Soundex`$_{GR}$

The `Soundex`$_{GR}$ algorithm and the procedures that are used by the algorithm are given in pseudocode in Alg. 1.

In the first part, we preprocess the word, applying to it syntax and grammar rules of the Greek language. Specifically, in `UnwrapConsonantBigrams`(*word*), we change common Greek *consonant digrams* with their equivalent, identically pronounced single letters. This is based on the substitutions shown in Table 2 (top part).

Then, in `TransformVowelsToConsonant`(*word*), we continue with identifying if the Greek Letter υ is acting as a vowel or as a consonant. This distinction needs to be made only if the letter before is α or ε, and the subsequent consonant falls in the category of Tables 4 and 5. For example,

αύξων [áfkson] (υ: consonant-φ),
αυτός [aftós] (υ: consonant-φ),
αυλή [avlí] (υ: consonant-β),
Εύξεινος [éfksinos] (υ: consonant-φ),
Εύδοξος [évdoksos] (υ: consonant-β),
ευεξία [eveksa] (υ: consonant-β).

Algorithm 1: $\texttt{Soundex}_{GR}$
Input: word
Output: An encoding of word

```
1: procedure $\texttt{Soundex}_{GR}$($word$)
2:     $w \leftarrow$UnwrapConsonantBigrams($word$)
3:     $w \leftarrow$TransformVowelsToConsonants($w$)
4:     $w \leftarrow$RemoveLast($w$)
5:     $w \leftarrow$GroupVowels($w$)
6:     $w \leftarrow$RemoveIntonation($w$)
7:     $code \leftarrow$SoundexEncode($w$)
8:     $code \leftarrow$RemoveDuplicates($code$)
9:     $code \leftarrow$TrimLength($code$, 4)
10:    return $code$
11: end procedure
```

```
1: procedure UNWRAPCONSONANTBIGRAMS($word$)
2:   for all digram in $word$ do
3:     if digram $= '\mu\pi'$ then Replace(digram, $'b'$)
4:     else if digram $= '\nu\tau'$ then Replace(digram, $'d'$)
5:     else if digram $\in \{'\gamma\kappa', '\gamma\gamma'\}$ then Replace(digram, $'g'$)
6:     else if digram $\in \{'\tau\sigma', '\tau\zeta'\}$ then Replace(digram, $'c'$)
7:     else if digram $\in \{'\pi\sigma', '\pi\varsigma'\}$ then Replace(digram, $'\psi'$)
8:     else if digram $\in \{'\kappa\sigma', '\kappa\varsigma'\}$ then Replace(digram, $'\xi'$)
9:     end if
10:  end for
11:  return $word$
12: end procedure
```

```
1: procedure TRANSFORMVOWELSTOCONSONANTS ($word$)
2:     for letter in $word$ do
3:         if letter $= '\upsilon'$ and previous $= '\alpha'$ or previous $= '\varepsilon'$ then
4:             if next is Silent (Table 5) then
5:                 Replace(letter, $\varphi$)
6:             else if next is Loud (Table 4) or next is Vowel then
7:                 Replace(letter, $= '\beta'$)
8:             end if
9:         end if
10:    end for
11:    return $word$
12: end procedure
```

```
1:  procedure GroupVowels (word)
2:      Replace(word, 'ά' → 'α')
3:      Replace(word, 'έ' → 'ε')
4:      Replace(word, 'ί' → 'ι')
5:      for all digram in word do
6:          if digram = 'αι' then Replace(digram, 'ε')
7:          else if digram ∈ {'ει', 'οι'} then Replace(digram, 'ι')
8:          else if digram = 'ου' then Replace(digram, 'ο')
9:          end if
10:     end for
11:     Replace(word, 'η', 'η', 'υ', 'ύ', 'ϋ', 'ΰ', 'ϊ', 'ΐ' → 'ι')
12:     Replace(word, 'ω', 'ώ', → 'ο')
13:     return word
14: end procedure
```

```
1:  procedure RemoveLast(word)
2:      if word.lastLetter = 'ς' or word.lastLetter = 'ν' then
3:          Replace(word.lastLetter, '')
4:      end if
5:  end procedure
```

After that, we remove letters ΄ς΄ or ΄ν΄ if they are the last letter of the word, as such letters do not add much value to the world.

In GroupVowels, we change common Greek *vowel digrams* with their equivalent, identically pronounced, single letters. This is based on the substitutions shown in Table 2 (bottom part).

In RemoveIntonation(*word*) (line 6), we remove possible remaining tones (if any); this is the last step of the word preprocessing phase.

In SoundexEncode(*word*) (line 7), we encode the word through the *letter-digit* pairs in Table 3. After translating the original word to a code, we *remove adjacent duplicate* digits in RemoveDuplicates(*code*) (line 8) and *trim length* to four characters or assign 0s to the end of the code if the code is smaller than four characters in trimLength(*code,4*) (line 9).

A few examples that show the outcome after each step of the algorithm are shown in Table 6.

To summarize the rules applied, Table 2 shows the 2-gram groups that produce *similar sounds* to a single letter, and as a result they are transformed to the corresponding single letter in the word preprocessing operation. Table 3 shows the complete set of phonetic buckets that are applied to the word as the final step in the encoding of the word. Table 4 shows the *Loud* category of the consonants in Greek which are used in order to identify if υ acts as a consonant, specifically a β, while Table 5 shows the Silent category of the consonants in Greek which are used in order to identify if υ acts as a consonant, specifically a φ. Note that the distinction to Loud and Silent concerns consonant phonemes. The silent ones contain those of Table 5 plus γκ, μπ, ντ, τζ; however, the last three are not needed for understanding the interpretation of υ, and this is the reason why they are not included in Table 5.

Table 3. *Soundex$_{GR}$* buckets

Soundex buckets	
Group	Code
β,φ,π,b	1
γ,χ	2
δ,τ,ϑ,d	3
ζ,σ,ξ,ψ,ς,c	4
ϰ,g	5
λ	6
μ,ν	7
ρ	8
α	9
ε	*
ο,ω	$
ι	@

Table 4. Loud consonants in Greek

Loud consonant
β
γ
δ
ζ
λ
μ
ν
ρ

3.4 *The algorithm* Soundex$_{GR}^{naive}$

For reasons of comparative evaluation, here we define another algorithm, that we call Soundex$_{GR}^{naive}$, that shares the same principles of the original Soundex algorithm, but without any word preprocessing before the encoding of the word. Specifically, the algorithm ignores vowels, has an encoding length of four characters, and does not encode the first letter. The only common aspect between this algorithm and Soundex$_{GR}$ is that it uses the same buckets from which the final encoding is generated, as shown in Table 3. Similarly to the original Soundex, we adopt the following steps:

Table 5. Silent consonants in Greek

Silent consonant
θ
κ
ξ
π
σ
ι
φ
χ

Table 6. Examples of *Soundex$_{GR}$* code generation, through different stages

Original Word	έμπειρος	νούς	ευάερος	δίαλλειμα	διάλυμα	αυλών	αυγό	αβγό	αυγουλάκια
UnwrapConsonant Bigrams	έbειρος	νούς	ευάερος	δίαλλειμα	διάλυμα	αυλών	αυγό	αβγό	αυγουλάκια
Transform Vowels To Consonants	έbειρος	νούς	εβάερος	δίαλλειμα	διάλυμα	αβλών	αβγό	αβγό	αβγουλάκια
RemoveLast	έbειρο	νού	εβάερο	δίαλλειμα	διάλυμα	αβλώ	αβγό	αβγό	αβγουλάκια
GroupVowels	εbιρο	νο	εβαερο	διαλλιμα	διαλιμα	αβλο	αβγο	αβγο	αβγολακια
SoundexEncode	ε1@8&	ν$	ε19*8*	δ@966@79	δ@96@79	α16$	α12$	α12$	α12$695@9
RemoveDuplicates	ε1@8&	ν$	ε19*8*	δ@9679	δ@9679	α16$	α12$	α12$	α12$695@9
TrimLength	ε1@8	ν$00	ε19*	δ@96	δ@96	α16$	α12$	α12$	α12$

(i) keep the first letter unencoded,
(ii) remove all occurrences of α, ε, ι, η, υ, ο, ω except when they appear as the first letter of the word,
(iii) replace consonants after the first letter as shown in Table 7,
(iv) remove adjacent duplicate digits,
(v) produce a code of the form Letter Digit Digit Digit by ignoring digits after the third one (if needed), or by appending zeros (if needed).

For example, this algorithm would encode αυγό to α200 and αβγό to α120, which are two identically sounded words, but with different encoding results. This evidences the superiority of Soundex$_{GR}$ in comparison to Soundex$_{GR}^{naive}$ (more such examples are included in Section 4.1).

3.5 *Phonetic matching with* Soundex$_{GR}^{comp}$

With Soundex$_{GR}$ we consider that two words w and w' *match*, denoted by $w \Leftrightarrow w'$, if they have the same code, that is, if Soundex$_{GR}(w)$ = Soundex$_{GR}(w')$. Analogously, with Soundex$_{GR}^{naive}$.

Table 7. Consonants Replacement in the *Soundex*$^{naive}_{GR}$

$\beta, \varphi, \pi \rightarrow 1$
$\gamma, \chi \rightarrow 2$
$\tau, \delta, \vartheta \rightarrow 3$
$\zeta, \sigma, \varsigma, \phi, \xi \rightarrow 4$
$\kappa \rightarrow 5$
$\lambda \rightarrow 6$
$\mu, \nu \rightarrow 7$
$\rho \rightarrow 8.$

In order to maintain both precision and recall levels as high as possible, here we introduce another variation for phonetic matching, that we call $\texttt{Soundex}^{comp}_{GR}$. The idea is to use both $\texttt{Soundex}_{GR}$ and $\texttt{Soundex}^{naive}_{GR}$ for keeping recall levels as high possible, without precision dropping. Specifically, this method uses both $\texttt{Soundex}_{GR}$ and $\texttt{Soundex}^{naive}_{GR}$ *in combination* during the matching process, that is, the query and the text are encoded with both the implementations, and if either one of them matches, then it is considered a match, that is:

$$w \Leftrightarrow w' \text{if } (\texttt{Soundex}_{\texttt{GR}}(w) = \texttt{Soundex}_{\texttt{GR}}(w')) \text{ OR } (\texttt{Soundex}^{naive}_{GR}(w) = \texttt{Soundex}^{naive}_{GR}(w'))$$

4. Evaluation

At first (in Section 4.1), we provide some indicative examples showcasing the merits of the codes and the differences between $\texttt{Soundex}^{naive}_{GR}$ and $\texttt{Soundex}_{GR}$. Then (in Section 4.2), we describe an evaluation collection that we have created containing datasets (`Dataset A` - `Dataset D`) with various types of errors and the metrics that we use for comparing the performance of various options (in Section 4.3). Subsequently (in Section 4.4), we report the evaluation results and discuss the related trade-offs (in Section 4.5). For further understanding of the performance of these codes, we also compare them with the lemmas produced by Greek stemmer (in Section 4.6), and we report measurements over a Greek dictionary (in Section 4.7). Furthermore (in Section 4.8), we provide and evaluate a method that yields a full phonetic transcription. In Section 4.9, we compare all methods, including the full phonemic transcription, plus edit distance-based methods, over an extended dataset of similarly sounded words `Dataset D`^{ext}, while in Section 4.10 we report the results of a series of experiments at different scales for understanding the factors that determine the optimal code length (`Dataset E` - `Dataset H`). Subsequently (in Section 4.11), we discuss efficiency, and finally (in Section 4.12) we discuss applicability and describe an application that showcases the benefits of $\texttt{Soundex}_{GR}$ for approximate matching.

An overview of the datasets that are used for evaluation purposes are given in Figure 1.

4.1 Indicative examples

Here, we provide a few indicative examples for understanding the behavior of $\texttt{Soundex}^{naive}_{GR}$ and $\texttt{Soundex}_{GR}$. Specifically, Table 8 provides examples where *both* $\texttt{Soundex}^{naive}_{GR}$ and $\texttt{Soundex}_{GR}$ tackle correctly various misspellings, that is, they assign the same code to all word variations.

Now Table 9 provides examples where $\texttt{Soundex}^{naive}_{GR}$ fails to assign the same code, while $\texttt{Soundex}_{GR}$ succeeds on providing the same code to all relevant word variations.

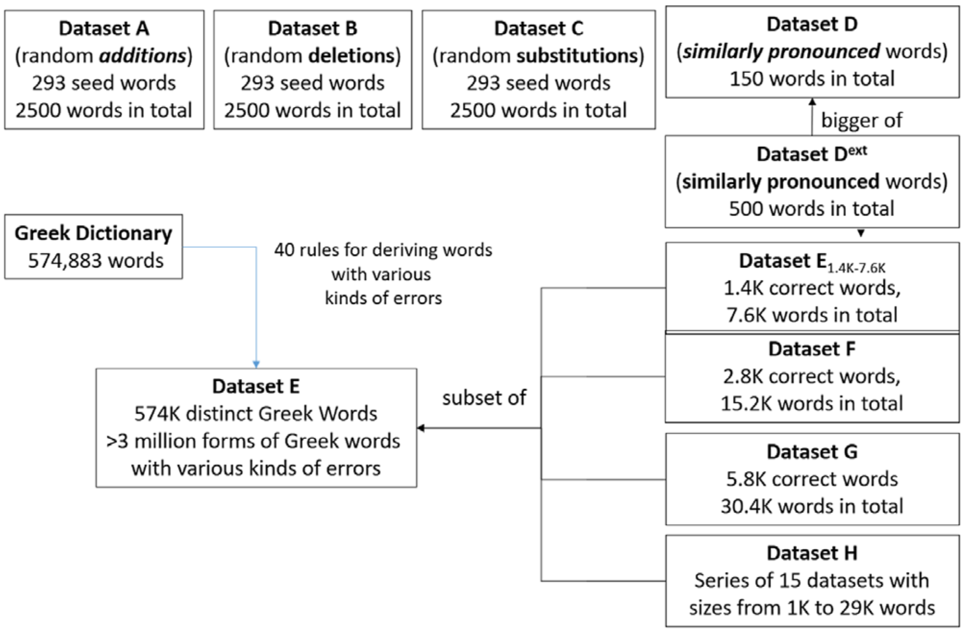


Figure 1. An overview of the datasets used for evaluation purposes.

4.2 Evaluation datasets (`Dataset A — Dataset D`*)*

There are various kinds of errors, for more see the extensive survey Kukich (1992), below we summarize the main ones. Human-generated misspellings sometimes tend to reflect typewriter keyboard adjacencies, for example, the substitution of “b” for “n” (in Greek β and ν). However, errors introduced by Optical Character Recognition (OCR) are more likely to be based on confusions due to featural similarities between letters (depending on the font), for example, the substitution of “D” for “O” (in Greek, we may encounter analogous problems with various groups of letters like Ο, Θ, Ω, as well as Α, Λ, Δ, and Ε, Σ and Υ, Ψ). We may also have the so-called typographic errors, for example, “spell” and “speel” (in greek ιππότης and ιποότης), where it is assumed that the writer knows the correct spelling but simply makes a motor coordination slip. There are also cognitive errors, for example, “receive” and “recieve” (in Greek φύλλο and φύλο each having a different meaning), due to a misconception or a lack of knowledge on the part of the writer. We can also encounter phonetic errors, for example, “abyss” and “abiss” (in Greek μήνυμα and μύνημα, δικλίδα and δικλείδα, καλύτερα and καλλίτερα) that are a special class of cognitive errors in which the writer substitutes a phonetically correct but orthographically incorrect sequence of letters for the intended word.

Apart from mistakes, there are words with more than one correct form, for example, αυγό and αβγό, and the same applies also for entity names, for example, the city of Heraklion is written as Ηράκλειο but also as Ηράκλειον, while the city of Athens is written Αθήνα but also Αθήναι.

Overall, according to Kukich (1992), nearly 80% of problems of misspelled words can be addressed either by addition of a single letter, or replacement of a single letter or swapping of letters. As the authors of Koneru *et al.* (2016) propose in their evaluation of various phonetic matching algorithms, we provide a similar evaluation collection for the Greek language that consists of datasets that contain words corresponding to various kinds of errors. Specifically, below we describe each of the four evaluation datasets that we have created. The set of words in each of these datasets contains verbs, nouns, adjectives, and proper names. The first three datasets,

Table 8. Indicative good examples for both $Soundex_{GR}^{naive}$ and $Soundex_{GR}$

word		$Soundex_{GR}^{naive}$	$Soundex_{GR}$
Θάλασσα	→	θ740	θ969
θάλλασα	→	θ740	θ969
θάλασα	→	θ740	θ969
μηνύμα	→	μ880	μ@7@
μύνημα	→	μ880	μ@7@
μίνιμα	→	μ880	μ@7@
μοίνειμα	→	μ880	μ@7@
τζατζίκι	→	τ434	c94@
τσατζίκι	→	τ434	c94@
τσατσίκι	→	τ434	c94@
κορονοιός	→	κ!84	κ8
κοροναιός	→	κ!84	κ8
Γιάννης	→	γ840	γ@97
Γιάνης	→	γ840	γ@97
Γιάνννης	→	γ840	γ@97
αναδιατάσσω	→	α833	α793
αναδιέταξα	→	α833	α793

`Dataset A`, `Dataset B`, and `Dataset C`, were created for checking how the algorithms behave in *various kinds of errors* (additions, deletions, and replacements) that can occur to a word, while last one, `Dataset D`, was created for evaluating *letter buckets*, that is, for testing the behavior of the matching in common errors.

In particular, `Dataset A` contains words produced by a single *random letter addition* to a random position in a word, for example, from the set of words αχρίδα, ωράριο, επιρρεπείς we produce words like αχρπίδα, ωρλάριο, επιτρρεπείς. Errors of this kind can happen by typing an extra keystroke. In `Dataset B`, the same procedure is used for *deletions*, that is, a letter is deleted from a random position, for the same set of words, for example, this dataset contains words like αχίδα, ωράρο, επιρρπείς. Again errors of this kind can happen during typing, that is, by a missing keystroke, or a typo (missing double letter). In `Dataset C`, we have random letter *substitution* in a random position, for example, in our example, we get words like αδρίδα, ωράριν, εψιρρεπείς. Again errors of this kind can happen during typing by one wrong keystroke (recall keyboard adjacencies, OCR errors, typographic, and cognitive errors).

Each of the above datasets contains 2500 words, generated from the same 293 unique words, that is, 7500 words in total. The generation of the erroneous words is random, that is, it does not consider any context or expected errors or typos. Finally, `Dataset D` contains 150 words comprising groups of similarly pronounced words, such as πολύ, πολλοί, πολλή, πωλεί and φύλλο, φίλο, φύλο, created manually. The motivation for creating this dataset was to capture some common errors, that is, frequently occurring spelling mistakes.

Table 9. Indicative examples where $Soundex^{naive}_{GR}$ fails while $Soundex_{GR}$ succeeds

word		$Soundex^{naive}_{GR}$	$Soundex_{GR}$
αυγό	→	α200	α12$
αβγό	→	α120	α12$
αυγολάκια	→	α276	α12$
αβλά	→	α120	α129
αυγά	→	α200	α129
έτοιμος	→	ε384	ε3@7
αίτημος	→	α384	ε3@7
αύξων	→	α480	α14$
άφξον	→	α148	α14$
εύδοξος	→	ε344	ε13$
εβδοξος	→	ε134	ε13$
θαύμα	→	θ800	θ917
θάβμα	→	θ180	θ917
θαυμαστικό	→	θ843	θ917
ξέρω	→	ξ!00	ξ*8$
κσαίρο	→	κ4!0	ξ*8$
οβελίας	→	ο174	ο1*6
ωβελύας	→	ω174	ο1*6
οβελίσκος	→	ο174	ο1*6
Βαγγέλης	→	β274	β95*
Βαγκέλης	→	β267	β95*
Βαγκαίλης	→	β267	β95*

4.3 Evaluation metrics

We shall use two basic metrics to evaluate the effectiveness of the algorithms, namely Precision and Recall. Precision is the portion of words that are retrieved and are relevant to the query, while Recall is the portion of relevant words that were retrieved, formally: $Precision = \frac{|(relevant) \cap (retrieved)|}{|(retrieved)|}$, $Recall = \frac{|(relevant) \cap (retrieved)|}{|(relevant)|}$. Let us now explain what "query", "retrieved", and "relevant" mean in our context. Each of the 293 unique words (of the first three datasets) is considered as query. For each such word *w*, the corresponding set of words in each dataset, that is, the words derived by making one modification, is considered as the set of relevant words.

For example, for the word ακρίδα, the set of relevant words is αγκρίδα, ακτρίδα, ακρφίδα (from `Dataset A`), ακρίδ, ακρία, κρίδα (from `Dataset B`), and ακίδα, ακρίφα, εκρίδα (from `Dataset C`). For each query word, the set of retrieved words is considered the set of *all words in all datasets* that have the same code. Then, for each dataset individually, we calculate the average

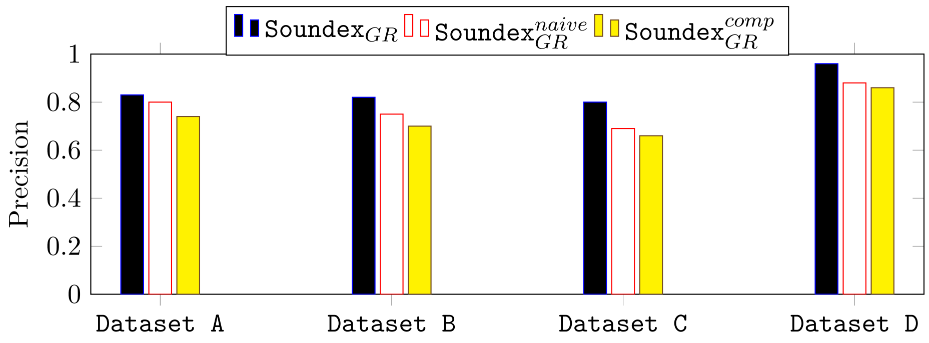


Figure 2. Precision levels for each collection.

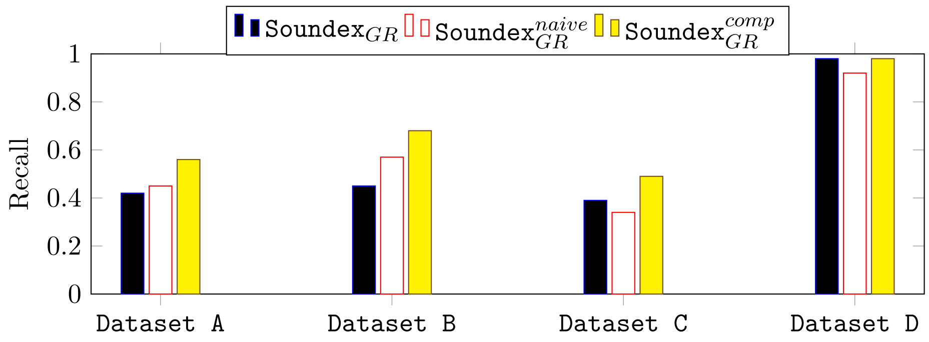


Figure 3. Recall levels for each collection.

Precision and average Recall, based on the Recall and Precision of each of the N queries, that is, $Precision_{avg} = \frac{\Sigma_{i=1}^{N} Precision_i}{N}$ and $Recall_{avg} = \frac{\Sigma_{i=1}^{N} Recall_i}{N}$.

4.4 Evaluation results over Dataset A – Dataset D

At first, we should note that if instead of applying any approximate algorithm, we apply exact match, then obviously we get Precision equal to 1, but the Recall is very low (around 0.1), as only one of the "relevant" words is fetched (of course the bigger the buckets of the group of words is in the evaluation datasets are, the lower the recall becomes). In Dataset A (the letter *addition* collection), Soundex$_{GR}$ achieved 0.83 precision and 0.42 recall, while Soundex$_{GR}^{naive}$ 0.80 and 0.45, respectively, while Soundex$_{GR}^{comp}$ achieved precision 0.74 and recall 0.56, as seen in Figure 2 (for precision) and Figure 3 (for recall).

In Dataset B (the letter *deletion* collection), Soundex$_{GR}^{naive}$ had a slight drop in precision to 0.75, and an increase in recall that reached to 0.57, while Soundex$_{GR}$ remained on the same level, with 0.82 and 0.45, respectively. Soundex$_{GR}^{comp}$ maintained a high precision level of 0.70 and achieved the higher recall 0.68, as seen in Figure 2 (precision) and Figure 3 (recall). The drop in the precision of Soundex$_{GR}^{naive}$ with the recall increasing is quite expected, since Soundex$_{GR}^{naive}$ ignores some letters and therefore it can handle better the deletion of a letter, while Soundex$_{GR}$ is more rigid to such errors.

In Dataset C (the letter *substitution* collection), Soundex$_{GR}^{naive}$ achieved precision 0.69 and recall 0.34. The lower scores are due to the more narrow set of phonetic rules. On the other hand, albeit a drop in the scores, the Soundex$_{GR}$ algorithm maintained the same level of score

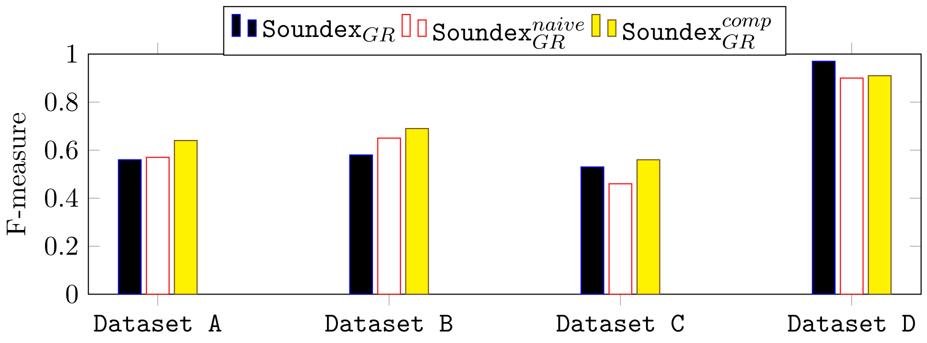


Figure 4. F-measure levels for each collection.

as in all three sets, with precision 0.80 and recall 0.39. In substitution, $\text{Soundex}_{GR}^{comp}$ did not manage to make a difference, as it combined the better results of Soundex_{GR} with the worse of $\text{Soundex}_{GR}^{naive}$, achieving precision 0.67 and recall 0.49, as seen in Figure 2 (precision) and Figure 3 (recall). Generally, the algorithms behave better when the error is ordinary to the common Greek Language, meaning that the word is still sounding as the correct one.

In `Dataset D`, the collection of *similarly pronounced* words, which comprises the main cases that a phonetic algorithm should be able to tackle both $\text{Soundex}_{GR}^{naive}$ and Soundex_{GR} got similar high scores, specifically $\text{Soundex}_{GR}^{naive}$ achieved precision 0.88 and recall 0.92, while the Soundex_{GR} achieved precision 0.96 and recall 0.98, as seen in Figure 2 (precision) and in Figure 3 (recall). The combination of the above algorithms, that is, $\text{Soundex}_{GR}^{comp}$, manages to maintain the high scores specifically precision 0.86 and recall 0.98, as its scores are dependent on the two implementations. These scores show that the buckets are sufficient, with Soundex_{GR} having slightly greater precision and recall score.

To sum up the results, we can see in Figure 4, that $\text{Soundex}_{GR}^{naive}$ achieves F-Score (note that F-Score, else called F-Measure, is the harmonic mean of precision and recall, that is, F-Score = $2\frac{Precision*Recall}{Precision+Recall}$) equal to 0.57, 0.65, 0.46, and 0.90 in `Dataset A`, `Dataset B`, `Dataset C`, and `Dataset D`, respectively. Soundex_{GR} achieves F-Score scores equal to 0.56, 0.58, 0.53, and 0.97, respectively, and the combination of the two $\text{Soundex}_{GR}^{comp}$ achieves 0.64, 0.69, 0.56, and 0.91, respectively, which shows that the $\text{Soundex}_{GR}^{comp}$ behaves better in general.

Both Soundex_{GR} and $\text{Soundex}_{GR}^{naive}$ achieved similar results. They work well when the error does not alter the generated code at a crucial point for the code. Both crucial points would be bellow four characters, and the error involving a consonant for $\text{Soundex}_{GR}^{naive}$, and a random, unexpected consonant or vowel that is not handled in the preprocessing of the word for the Soundex_{GR}. Since $\text{Soundex}_{GR}^{comp}$ includes both implementations in the retrieval process, it shares the same issues but manages to have higher recall values while not sacrificing greatly in precision. Using both codes can increase recall levels by 0.05 to 0.20, while the precision suffers a drop from 0.10 to 0.20, comparing to Soundex_{GR}. The algorithms work well in retrieving words, if the error in a word is based on the same phonetic rules (of Table 3) or are caught in the preprocessing stage, when we make both the query and the text as mispronounced as possible, especially Soundex_{GR}. For example, for a query like `κατεβαίνω`, it would correctly retrieve `κατεβένω`, `καταιβαίνω`, `κατεβαίνο`, `κατεμπένο`, `κατεβενο`, `κατεββαίνω` but not `κατβαίνω`, `κτεβαίνω`, `ρατεβαίνω`. This is the case because, a single letter addition/deletion/substitution will change the Soundex code, and Soundex does not have a similarity metric in the comparison process.

Table 10. Average F-Score (over Dataset A, Dataset B, Dataset C, Dataset D) for different lengths of *Soundex$_{GR}$*

Soundex$_{GR}$	F-Score				
Length	Dataset A	Dataset B	Dataset C	Dataset D	Avg F-Score
1	0.11	0.12	0.11	0.41	0.18
2	0.37	0.36	0.36	0.72	0.45
3	**0.57**	**0.60**	**0.56**	0.90	0.65
4	0.56	0.58	0.53	0.97	**0.66**
5	0.47	0.51	0.42	**0.98**	0.59
6	0.40	0.45	0.35	**0.98**	0.54
7	0.36	0.43	0.29	**0.98**	0.51
8	0.35	0.42	0.28	**0.98**	0.50
9	0.34	0.41	0.27	**0.98**	0.50
10	0.34	0.41	0.26	**0.98**	0.49
15	0.34	0.41	0.26	**0.98**	0.49

4.5 Discussion of the revealed trade-offs as Regards the Length of the Codes (over Dataset A — Dataset D)

While testing the algorithm, we observed that simple changes affect the achieved precision and recall. For instance, changing the *length* of the encoding of Soundex$_{GR}$ from 4 to 6 would greatly improve precision from 0.80 to over 0.90, while dropping recall from 0.40–0.45 to 0.25–0.30. Although Soundex algorithms are used mainly in context where recall matters the most, it is wise to choose the algorithm that suits better the requirements of the application context, that is, whether emphasis should be given to precision or recall. We also noticed that by leaving the first letter unencoded, as the original Soundex, we get a slight increase in the precision (by 0.05–0.10), and a decrease in the recall by 0.05. Finally, splitting all the letters to more categories would also increase precision and decrease recall.

To better understand how the length of the Soundex$_{GR}$ codes affects the obtained F-Score, we computed the F-Score over all datasets for code length starting from 1 up to 10, and the length 15. The results are shown in Table 10. The rightmost column shows the average F-Score over each of the four datasets. We can see that length 4 yields the best average F-Score.

To better understand how Precision and Recall are affected by the length of the code, Figure 5 shows for each dataset the Precision, Recall, and F-Score for each length from 1 to 10. In the datasets that correspond to various kinds of errors, that is, in Dataset A (the letter *addition* collection), Dataset B (the letter *deletion* collection), and Dataset C (the letter *substitution* collection), we can see clearly that as the code length increases, the precision increases but the recall decreases. The code length where the F-Score is maximized in these three datasets is 3. In Dataset D (the collection of *similarly pronounced* words), we can see that as the length increases, the precision increases as well, reaching its maximum at length 5. The recall level does not decrease as the code length increases (as it happens in the previous three datasets) because, even with big code length, the set of all relevant words are those that sound the same and all of them are retrieved because Soundex$_{GR}$ succeeds in assigning them the same code. In this dataset, the length that maximizes F-Score is 5 and any bigger length.

More experiments on the selection of the codes' length are given and analyzed in Section 4.10.

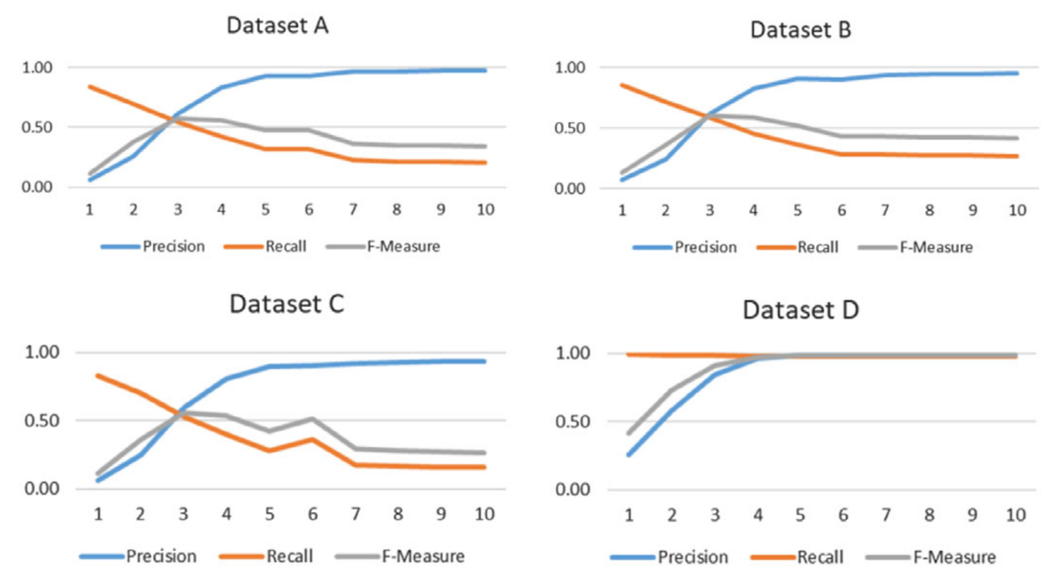


Figure 5. Precision, Recall, and F-Score evaluation metrics on `Dataset A` (top left), `Dataset B` (top right), `Dataset C` (bottom left), and `Dataset D` (bottom right) for *Soundex*$_{GR}$ code lengths 1 to 10.

4.6 Comparison with stemming

Apart from comparing the various variations of Soundex$_{GR}$ we decided to compare the grouping of words that it is obtained through Soundex$_{GR}$, with the grouping that it is obtained by a Stemmer for the Greek language. In general, *stemming* refers to the process of reducing inflected (or derived) words to their word base or root form. Note that the stem is not necessarily the morphological root of the word in the sense that if two related words map to the same step, then even this stem is not a valid root[a], and it is sufficient for the task of matching and retrieval. Consequently, the strong point of using a stemmer for the problem of matching is that it can successfully identify morphological variations of the same word, and thus it can match word forms that are orthographically and phonetically quite different; however, the weak point of using a stemmer for matching is that it cannot tackle typos (stemmers have not been designed for overcoming typing mistakes) and cannot be applied to named entities (persons, addresses, places, companies, etc).

We used one stemmer of the Greek language, specifically the Mitos Greek Stemmer (Karamaroudis and Markidakis 2006) described in Papadakos *et al.* (2008) and applied it over the same datasets. The results for Precision, Recall, and F-Score are shown in Figure 6, 7, and 8 respectively.

We can see that stemming has higher precision (as expected), that is, if two words have the same stem then with high probability they belong to same category of words; however, the recall is very low (as expected), since it cannot tackle misspellings that sound the same. Consequently, stemming has a poor F-Score in comparison to Soundex$_{GR}$; only in `Dataset C` stemming has comparable performance (with performance similar to that of Soundex$_{GR}^{naive}$). Overall, Soundex$_{GR}$ is significantly better for the problem at hand, in comparison to using an ordinary stemmer.

Finally, we should note that we tried also the scenario where we first apply stemming and then apply the Soundex (over the stemmed words); however, the results were worse.

[a]as it happens for the English language with the Porter stemmer https://tartarus.org/martin/PorterStemmer/ for the English language

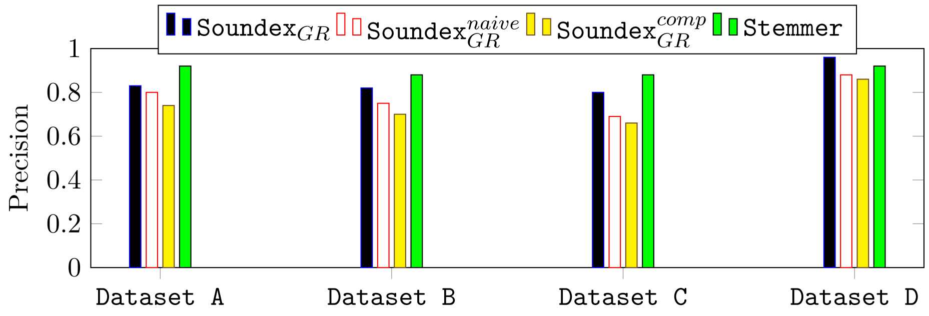


Figure 6. Precision levels for each collection (also for stemming).

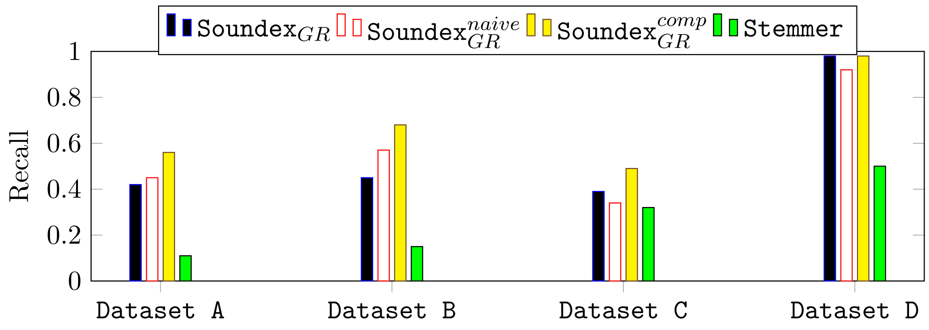


Figure 7. Recall levels for each collection (also for stemming).

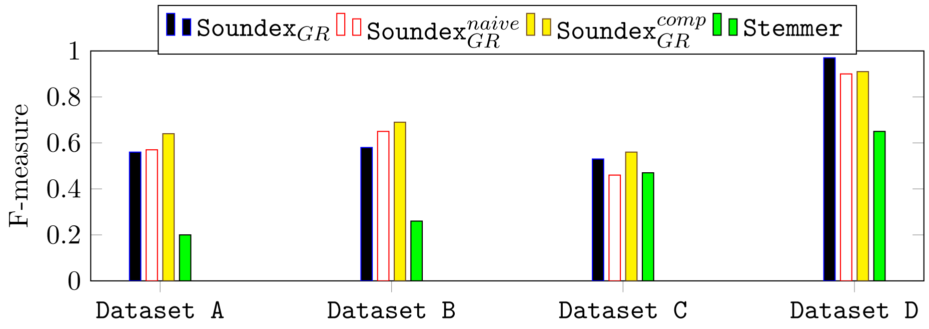


Figure 8. F-measure levels for each collection (also for stemming).

More comparative experiments with stemmer-based matching are given in Section 4.9, as well as in the series of experiments described in Section 4.10.

4.7 Measurements over a Greek dictionary

A dictionary is not a kind of dataset for evaluating phonetic algorithms, since it neither contains misspelled words nor persons' last names, location names, etc. However, we decided to perform some measurements for getting one idea about the distribution of codes (and for measuring efficiency). For this purpose, we used the WinEdt Unicode dictionary for Greek[b]. That dictionary

[b]Greek WinEdt Unicode Dictionary, version 2008-10-03, downloaded April 26, 2020, size 2,089 KB, http://www.winedt.org/dict.html.

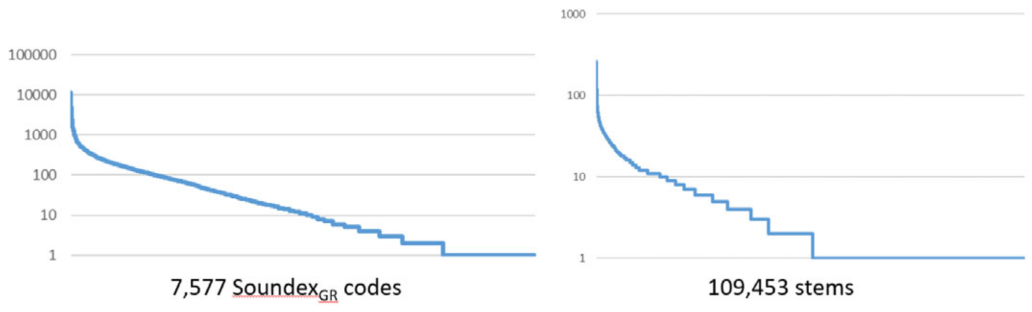


Figure 9. Frequency of *Soundex$_{GR}$* codes (left), and lemmas of the stemmer (right) over the dictionary.

contains Greek words and their morphological variations, as well as fist names and acronyms, for example, it contains Γιάννης, Γιάννη, ΑΕΙ. It is actually a list of words, and in total it contains more than half a million Greek words (specifically 574,883). The total number of characters of these words is 6,279,813; hence, the average word size is 10.92 characters and the smallest word(s) have a length 3, while the bigger one has a length 27 (στρογγυλοκουλουριαζόντουσαν).

Since the average number of characters per word is 10.92, while each Soundex$_{GR}$ code comprises four characters, the size of these codes correspond to the 36% of the size of the original dictionary (or we will have 36% increase in the dictionary size if we decide to store also the Soundex$_{GR}$ code for each word). Using the stemmer that we mentioned in Section 4.6, the average stem size is 7.46. That means that the size of these stems correspond to the 68% of the size of the original dictionary (or we will have 68% increase in the dictionary size if we decide to store also the stem for each word).

The number of distinct Soundex$_{GR}$ codes is 7577, that is, in average each code corresponds to 574,883/7577 = 75.87 words. The number of distinct stems is 109,453, that is, each stem corresponds to 574,883/ 109,453 = 5,25 words. In comparison to Soundex$_{GR}$, the number of lemmas is 109,4530/7577 = 14.44 times more than the number of Soundex$_{GR}$ codes.

The distribution of neither Soundex$_{GR}$ codes, nor stems, is uniform, as expected. There are codes with only one word, while the more "populated" code corresponds to 11,681 words (corresponding to words starting from κατα, a frequent prefix in Greek). Analogously, the min number of words per stem is 1, while the max number of words per stem is 257 (corresponding to the lemma ταξ). The distributions of the frequencies of Soundex$_{GR}$ codes, and stemmer lemmas, over the distionary, are shown in Figure 9, where both Y-axes (of the left and right plot) are in log scale. The 10 more frequent codes are shown in Table 11, while the 10 more frequent stems at Table 12.

Of course, and based on the task at hand, one might decide to use longer Soundex$_{GR}$ codes if he wants to improve precision over recall, as discussed in Section 4.5.

4.8 Other variations: Full phonemic transcription

It is not hard to see that the same rules, with small changes, can be used for deriving the full *phonemic* transcription of a Greek word. With "phonemes," we refer to the mental categories that a speaker uses, rather than the actual spoken variants of those phonemes that are produced in the context of a particular word (note that phonetic transcription specifies the finer details of how sounds are actually made).

Specifically, we can use only the following three steps of Alg. 1:

$w \leftarrow$UnwrapConsonantBigrams($word$)

$w \leftarrow$TransformVowelsToConsonants(w)

$w \leftarrow$GroupVowels(w)

Table 11. More frequent *Soundex*$_{GR}$ codes

code	frequency
κ939	11681
π989	8207
π84	5396
π*8@	4756
π$6@	4399
ξ979	3953
α1$3	3949
μ*39	3933
α*1$4	3857
Π8$3	3595

Table 12. More frequent stems

code	frequency
ταξ	257
μασ	235
αρξ	233
κοψ	212
πασ	209
βασ	208
ποα	194
γραψ	190
παξ	176
ορξ	159

The rest steps of Alg. 1 are not needed, that is, we skip the step of removing last chars (*RemoveLast*), the step of encoding (*SoundexEncode*), and the step of duplicate elimination (*RemoveDuplicates*).

With the above three steps, the changes that are required for producing a full phonetic transcription of Greek words are minimal. The first change is that in *GroupVowels(w)* the grouping is a bit different, specifically we group "ου" to"u" (instead of "o"). The second change is that instead of mapping both "τσ' and "τζ' to "c" we map the first to "ts" and the second to "dz". Finally, instead of using greek letters for the phonetic transcription we can use latin letters whenever possible, in any case the selection of the characters in the phonetic transcription does not affect the matching process. A few examples are given in Figure 10:

word		$\texttt{Soundex}^{naive}_{GR}$	$\texttt{Soundex}_{GR}$	Phonemic Transcription
αυγό	→	α200	α12$	ανγο
αβγό	→	α120	α12$	ανγο
εύδοξος	→	ε344	ε13$	evδoξος
εβδοξος	→	ε134	ε13$	evδoξος
λιανοτράγουδα	→	λ83!	λ@97	lianotraγuδa
στρογγυλοκουλουριαζόντουσαν	→	σ3!2	σ38$	strogilokuluriaζodusan

Figure 10. Indicative examples of full phonemic transcription.

We have implemented the above version, and it is included in the public release of the SoundexGR family of algorithms (described in Section 4.11). Another important question is how the exact phonetic (phonemic) transcription would behave in the evaluation datasets (described in Section 4.2). The results are not that good, specifically:

in `Dataset A` (the letter *addition* collection) we got F-Score = 0.17,
in `Dataset B` (the letter *deletion* collection) we got F-Score = 0.31,
in `Dataset C` (the letter *substitution* collection) we got F-Score = 0.23, and
in `Dataset D` (the collection of *similarly pronounced* words) we got F-Score = 0.93. We observe that full phonetic transcription behaves well only in `Dataset D` achieving F-Score 0.93; however, that score is lower than 0.97 that is achieved by by $\texttt{Soundex}^{naive}_{GR}$. As expected, in the rest evaluation datasets, the exact phonetic transcription behaves much worse since it cannot tackle the cases of letter additions, deletions, and substitutions.

Overall, the average F-Score across all evaluation datasets of $\texttt{Soundex}_{GR}$ for length 4 is equal to 0.66 (as shown in Table 5), while the average F-Score across all evaluation datasets of full phonemic transcription is 0.41 (=(0.17+0.31+0.23+0.93)/4).

Additional experiments with matching using full phonemic transcription are given in Section 4.9, and in the series of experiments described in Section 4.10.

4.9 Comparing all variations over `Dataset D`^{ext}

To provide an overview of the effectiveness of the aforementioned methods, we decided to prepare an extended version of `Dataset D` for containing more variations for each word. The derived dataset, denoted by `Dataset D`^{ext}, contains in total 500 words, in particular it contains 125 words in their orthographically correct form plus 3 misspellings for each one of these. All of the misspellings sound the same with the correct one. We have tried to include words that are frequently misspelled as well as typographic errors that do not, however, change the way they would sound. An excerpt of this dataset is shown in Figure 11.

Over this dataset, we evaluated all aforementioned methods, plus some more, 10 in total methods, in particular exact match, $\texttt{Soundex}^{naive}_{GR}$, $\texttt{Soundex}_{GR}$, $\texttt{Soundex}^{comp}_{GR}$, stemming (as described in Section 4.6), $\texttt{Soundex}_{GR}$ over the results of stemming, full phonemic transcription (as described in Section 4.8), and matching based on the edit distance Levenshtein (1966) with tolerance K ranging from 1 to 3. For instance, edit distance with $K = 2$ means that two words match if their edit distance is less than or equal to 2. The code length for $\texttt{Soundex}^{naive}_{GR}$, $\texttt{Soundex}_{GR}$, and $\texttt{Soundex}^{comp}_{GR}$ was equal to 4. The results are shown in Table 13, where the highest values of Precision, Recall, and F-Score are written in bold. By inspecting the values, we can understand the behavior of these methods, and we can see that $\texttt{Soundex}_{GR}$ achieves the highest F-Score (0.97).

Table 13. Evaluating 10 matching methods over *Dataset* D^{ext}

	Method	Precision	Recall	F-Score
1	exactMatch	**1.0**	0.25	0.40
2	$Soundex_{GR}$	0.95	**0.99**	**0.97**
3	$Soundex_{GR}^{naive}$	0.92	0.91	0.91
4	$Soundex_{GR}^{comp}$	0.88	**0.99**	0.93
5	Stemmer	0.94	0.30	0.46
6	$Soundex_{GR}$ over Stemmer	0.85	0.79	0.82
7	Full phonemic transciption	**1.0**	0.66	0.80
8	Edit Distance ≤ 1	0.97	0.58	0.73
9	Edit Distance ≤ 2	0.78	0.84	0.81
10	Edit Distance ≤ 3	0.52	0.93	0.67

βύσσινο, βύσινο, βύσυνο, βύσιννο
διάλλειμα, διάλυμα, διάλοιμα, διάλειμα
παλίρροια, παλίροια, παλίρια, παλείρεια
παράλειψη, παράληψη, παράλιψη, παράλειψψη
πλημμύρα, πλημύρα, πλημίρρα, πλοιμοιρα
ωράριο, οράριο, ωράρειο, οράροιο

Figure 11. Excerpt from `Dataset D`^{ext}.

4.10 Experiments at different scales—On selecting the length of the codes (over `Dataset E - Dataset H`***)***

In Section 4.5, we have seen that the length 4 yields the best average F-Score over the four evaluation datasets. Questions that arise are: Does the optimal length depend on the size of the dataset? Should we use shorter codes in smaller datasets, and larger codes in larger collections? One approach for tackling these questions is to make the experiments (like those reported in Table 10) but instead of considering the entire evaluation datasets, to consider only parts of these datasets starting from very small parts and reaching to the entire evaluation datasets. For this purpose, we performed experiments after having limited the number of words to be considered from each dataset, starting from 200 words up to 2000 words with increment step equal to 200.

For each such dataset size, we have evaluated $\texttt{Soundex}_{GR}$ code lengths starting from 2 up to 12. The experimental results, as regards average F-Score, are shown in Figure 12(top plot). The left Y-axis corresponds to the code length (from 2 to 12), while the right Y-axis corresponds to the average F-Score (across the four evaluation dataset parts). The X-axis shows the dataset sizes (from 200 to 2000 words with step equal to 200), and for each such size the X-axis has 11 ticks each corresponding to one code length (from 2 to 12).

Figure 12(top plot) reveals the following general pattern: as the code length increases, the F-Score increases reaching a peak around 0.7 (usually for code length 3 or 4) and then it is decreased and ends up to 0.5. Figure 12 (middle and bottom plot) shows the average Precision and average Recall that helps us to explain the distribution of the average F-Score. From these measurements, we could say that the size of the dataset is not very decisive (at least for the considered sizes in this experiment, i.e., for 200 to 2000), since we can see that the size of the dataset does

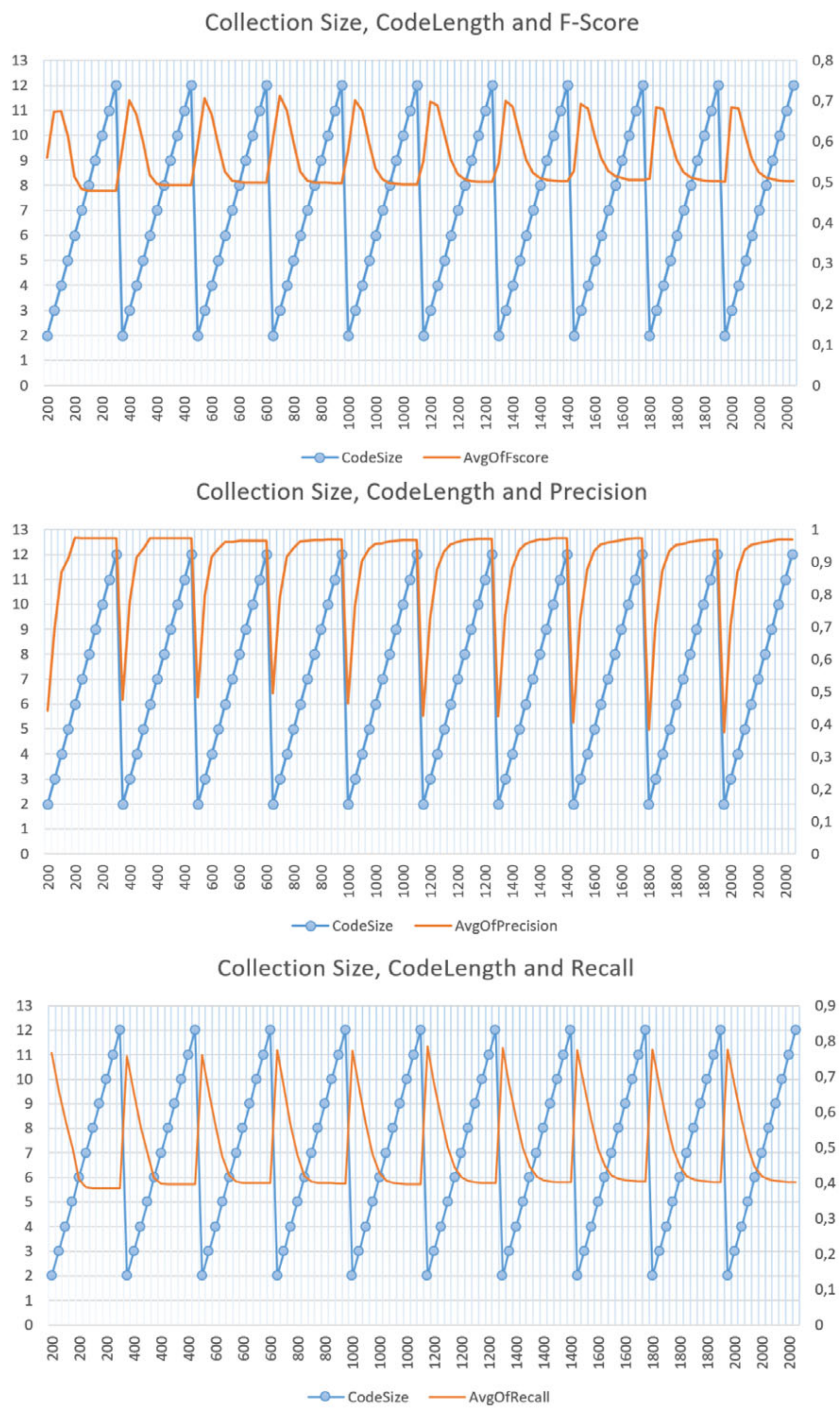


Figure 12. Average F-Score (top), Precision (middle), and Recall (bottom) as a function of code length (left Y-axis, blue dots) and dataset size (X-axis) of *Soundex*$_{GR}$ in `Dataset A`, `Dataset B`, `Dataset C`, and `Dataset D`.

- ϑολά, ϑωλά, ϑολλά
- Χάρης, Χάρυς, Χάρρνς
- αποπήρε, αποπύρε, αποπήραι, απωπήρε, αππoπήρε
- επιρρεπώς, επυρρεπώς, αιπυρρεπώς, επυρρεπός, εππυρεπός
- λιπόψυχης, λιπόψυχυς, λιπόψηχυς, λυπόψηχυς, λυπώψηχυς, λυππόπσηχυς
- προσυπογραφόμουν, προσηπογραφόμουν, πρωσηπογραυόμουν, πρωσηπογραυώμουν, προσηπογραυ-ώμουν, πρροσηπογραυόμουν
- ετεροδημότη, ετεροδυμότη, ετεροδημότη, ετεροδημότει, αιτεροδημότοι, ετερωδημότοι, ετερωδημώτοι, ετεροδημώτοι, εττεροδημότοι
- δεντροφυτευόταν, δαιντροφητευόταν, δεντροφητεβόταν, δεντροφητεφόταν, δεντρωφητευόταν, δεντρω-φητευώταν, δεντροφητευώταν, δεννтροφητευόταν

Figure 13. Excerpt from *Dataset* $E_{1.4K-7.6K}$.

not affect significantly the F-Score. It is not hard to see that it is not only the size of the dataset that matters, but also the length of the words, a quantity that does not depend on the dataset size: even in small datasets too, short codes or too long codes harm the F-Score that we achieve and this is evidenced by the measurements, that is, through the low F-Score values that we get for very short and very long codes in Figure 12(top plot). To test this hypothesis, and further understand what affects performance, we designed the experiment that follows.

Datasets with bigger word size variations. By exploiting the experience from creating (manually) the `Dataset D`^{ext}, we decided to use the dictionary of Greek words (mentioned in Section 4.7 that contains 574,883 distinct words) for producing larger datasets for further evaluation and experimentation related to the size of the codes. For each word of that dictionary, we produce a bucket that contains variations of the word with various kinds of errors. We decided to include words that contain more than one errors, not only because there are many frequent misspellings that contain more than one error, for example, μύνημα instead of μήνυμα, θάλλασα instead of θάλασσα, but also for evaluating cases that cannot be captured easily by the edit distance. Therefore, we have included various errors that do not affect the way the word sounds, so the emphasis is given on orthographic errors.

Specifically, for producing such errors, we have created around 40 rules for capturing various cases. Most of them are replacement rules, with conditions on the characters that should not appear before or after the character to be replaced. For instance, the rule Rule ({ο,α,ε},ι,υ,-) replaces ι with υ only if the letter before ι is not one of ο,α,ε, since in that case we have a diphthong and such an error would not be common. Analogously, the rule Rule (-,ο,ω,{υ,ύ,ι,ί}) replaces ο with ω only if the character after ο is not one of the lists, since in that case we have a diphthong too. The set of rules is not supposed to produce all possible errors, but they can capture pretty well various kind of common errors; therefore, the variations they produce can be used for the evaluation of approximate matching. To ensure that for each word (also for the very small ones) we have at least one misspelled word, we have included one rule that doubles a middle consonant. Let call this dataset `Dataset E`.

The words in the original dictionary are ordered by their size. To create a dataset that covers all word sizes we used step 400, that is, we peek one word every 400 words of the dictionary. The resulting dataset, that we will denote by `Dataset` $E_{1.4K-7.6K}$, has 1438 distinct correct words 7608 words in total, and the average size of the blocks is 5.29, that is, in average the dataset contains more than four misspellings per word. A small excerpt from the produced dataset is shown in Figure 13.

Over this dataset, denoted by `Dataset` $E_{1.4K-7.6K}$, we run the experiments and the results are shown in Table 14. At first, we observe that exact match achieves F-Score 0.37, stemming 0.40, while full phonemic transcription 0.86. Edit distance achieves its maximum F-Score, that is, 0.9,

Table 14. Evaluating 10 methods over *Dataset* $E_{1.4K-7.6K}$

	Method	Precision	Recall	F-Score
1	exactMatch	**1.0**	0.23	0.37
2-i	$Soundex_{GR}$ (4)	0.63	0.96	0.76
3-i	$Soundex_{GR}^{naive}$ (4)	0.74	0.89	0.80
4-i	$Soundex_{GR}^{comp}$ (4)	0.55	0.98	0.70
2-ii	$Soundex_{GR}$ (5)	0.82	0.96	0.89
3-ii	$Soundex_{GR}^{naive}$ (5)	0.91	0.88	0.90
4-ii	$Soundex_{GR}^{comp}$ (5)	0.78	0.98	0.87
2-iii	$Soundex_{GR}$ (6)	0.93	0.96	0.94
3-iii	$Soundex_{GR}^{naive}$ (6)	0.96	0.87	0.91
4-iii	$Soundex_{GR}^{comp}$ (6)	0.90	0.98	0.94
2-iv	$Soundex_{GR}$ (7)	0.97	0.95	0.96
3-iv	$Soundex_{GR}^{naive}$ (7)	0.98	0.88	0.92
4-iv	$Soundex_{GR}^{comp}$ (7)	0.95	0.98	**0.97**
2-v	$Soundex_{GR}$ (8)	0.98	0.95	**0.97**
3-v	$Soundex_{GR}^{naive}$ (8)	0.98	0.87	0.92
4-v	$Soundex_{GR}^{comp}$ (8)	0.97	0.98	**0.97**
2-vi	$Soundex_{GR}$ (9)	0.99	0.95	**0.97**
3-vi	$Soundex_{GR}^{naive}$ (9)	0.98	0.87	0.92
4-vi	$Soundex_{GR}^{comp}$ (9)	0.97	0.98	**0.97**
2-vii	$Soundex_{GR}$ (10)	0.99	0.95	**0.97**
3-vii	$Soundex_{GR}^{naive}$ (10)	0.98	0.87	0.92
4-vii	$Soundex_{GR}^{comp}$ (10)	0.97	0.98	**$\underline{0.98}$**
2-viii	$Soundex_{GR}$ (11)	0.99	0.95	**0.97**
3-viii	$Soundex_{GR}^{naive}$ (11)	0.98	0.87	0.92
4-viii	$Soundex_{GR}^{comp}$ (11)	0.97	0.98	**0.98**
5	Stemmer	0.98	0.25	0.40
6	Full phonemic transciption	0.99	0.75	0.86
7	Edit Distance ≤ 1	0.99	0.44	0.60
8	Edit Distance ≤ 2	0.99	0.69	0.82
9	Edit Distance ≤ 3	0.95	0.87	0.90
10	Edit Distance ≤ 4	0.80	0.95	0.87

Table 15. Evaluating 10 methods over *Dataset* $F_{2.8K-15.2K}$

	Method	Precision	Recall	F-Score
1	exactMatch	**1.0**	0.22	0.37
2-i	$Soundex_{GR}$ (4)	0.5	0.96	0.65
3-i	$Soundex_{GR}^{naive}$ (4)	0.61	0.89	0.72
4-i	$Soundex_{GR}^{comp}$ (4)	0.40	0.98	0.57
2-ii	$Soundex_{GR}$ (12)	0.99	0.95	**0.97**
3-ii	$Soundex_{GR}^{naive}$ (12)	0.96	0.87	0.91
4-ii	$Soundex_{GR}^{comp}$ (12)	0.96	0.98	**0.97**
5	Stemmer	0.96	0.25	0.40
6	Full phonemic transciption	0.99	0.75	0.85
7	Edit Distance ≤ 1	0.99	0.43	0.60
8	Edit Distance ≤ 2	0.98	0.69	0.81
9	Edit Distance ≤ 3	0.91	0.86	0.89
10	Edit Distance ≤ 4	0.71	0.95	0.81

with $K \leq 3$. Notice that $\texttt{Soundex}_{GR}$ is better than all the above options for any code length equal or greater than 6. The optimal F-Score is, that is, 0.98, is achieved with $\texttt{Soundex}_{GR}^{comp}$ and code length equal to 10. This length is longer than what we expected; however, this can be explained by the fact that the dictionary contains a lot of big words.

To produce a larger dataset, we reduced the step to 200 and we produced $\texttt{Dataset F}_{2.8K-15.2K}$ that contains 2875 correct words and 15,297 total words (average bucket size 5.32). The results of the experiments are shown in Table 15. We observe a slight drop in precision and F-Score for length 4; however $\texttt{Soundex}_{GR}$ with code length equal to 12 preserves the very high F-Score (0.97).

To produce an even larger dataset, we further reduced the step to 100 and produced $\texttt{Dataset G}_{5.7K-30.4K}$ that contains 5749 correct words and 30,824 words in total (average bucket size 5.36). The results of the experiments and the results are shown in Table 16. We observe a further drop in precision and F-Score for length 4; however, for code length equal to 12, $\texttt{Soundex}_{GR}$ preserves the very high F-Score (0.97).

The previous datasets ($\texttt{Dataset E}_{1.4K-7.6K}$ – $\texttt{Dataset G}_{5.7K-30.4K}$), which were derived by picking words from the beginning up to the end of the dictionary, covered the entire spectrum of word lengths. However, longer words are less frequent; therefore, it is sensible to make experiments starting from the beginning and without gaps, for considering all short- and medium-sized words, which are expected to contain the frequent ones. The resulting dataset is probably harder for matching, not only because there are many small words making precision hard to achieve, but also because many morphological variations of the included words will be included (since Step 1 was used), so it is more challenging to achieve high precision. For this reason, we performed experiments of $\texttt{Soundex}_{GR}$ for all code lengths from 2 up to 12 for dataset sizes starting from 1000 words to 29,000 words with dataset increment step 2000 (words, not rows). The resulting series of 15 datasets contain letters with words up to 6 letters.

The results are given in Figure 14. Notice that the right vertical axes start from 0.5 for F-Score, 0.3 for Precision, 0.8 for Recall, to make more evident the differences. In Figure 14(top plot), we observe that Recall is not essentially affected by neither dataset size nor code length. In Figure 14 (middle plot), we observe that (as expected) the Precision is lower and it is affected by the size

Table 16. Evaluating 10 methods over *Dataset* $G_{5.7K-30.4K}$

	Method	Precision	Recall	F-Score
1	exactMatch	**1.0**	0.22	0.36
2-i	$Soundex_{GR}$ (4)	0.36	0.96	0.52
3-i	$Soundex_{GR}^{naive}$ (4)	0.46	0.88	0.61
4-i	$Soundex_{GR}^{comp}$ (4)	0.26	0.98	0.41
2-ii	$Soundex_{GR}$ (12)	0.99	0.95	**0.97**
3-ii	$Soundex_{GR}^{naive}$ (12)	0.93	0.87	0.90
4-ii	$Soundex_{GR}^{comp}$ (12)	0.92	0.98	0.95
5	Stemmer	0.92	0.24	0.38
6	Full phonemic transciption	0.99	0.75	0.85
7	Edit Distance ≤ 1	0.96	0.43	0.60
8	Edit Distance ≤ 2	0.97	0.68	0.80
9	Edit Distance ≤ 3	0.86	0.86	0.86
10	Edit Distance ≤ 4	0.61	0.95	0.74

of the collection. In Figure 14 (bottom plot), we observe that F-Score is affected by the size of the collection (i.e., it decreases as the dataset size increases) but achieves 0.7 for code lenghs ≥ 8. In general, we observe (as expected) that in this series of datasets that contains small words, the F-Score is lower than what in Dataset $G_{5.7K-30.4K}$. This evidences that not only the size of the vocabulary and the kind of errors but also the size of the words affect the effectiveness of matching.

Synopsis and general remarks. Figure 15 illustrates the main results, that is, it shows each dataset and its characteristics, as well as the best F-Scores obtained by $\texttt{Soundex}_{GR}$ and other matching methods.

A few general remarks follow:

- The bigger the collection is, and the longer words it contains, the longer the codes should be (to preserve precision). The same is true for the tolerance of edit distance-based matching. In a context where retrieval of high precision is required (e.g., in the retrieval of user comments within a voice-based conversational interaction, as in Dimitrakis *et al.* (2018)), longer codes can be selected, while in an application context where recall is more important (e.g., in patent search), shorter ones could be more appropriate. The performance also depends on the kind of errors that expect and their relative percentage (e.g., long codes are good if we have several orthographic errors, not random errors).
- If one wants to select the best option in a particular application setting, apart from the above analysis, one can perform ad hoc experiments, and for this reason the code for running the aforementioned experiments with various sizes of codes has been made publicly available. Moreover, and to facilitate comparative results, we have uploaded the full dataset that contains 574,883 distinct Greek words and 4.32 misspellings per word in average, in total more than 3 million forms of Greek words (3,063,143) at Tzitzikas (2021).

4.11 Implementation and efficiency

As regards efficiency, using a machine with 1.8 GHz i7, 4MB cache, and 16 GB of RAM, $\texttt{Soundex}_{GR}$ encodes the words of each set of 2500 words in 2.5 s, meaning each word takes 1

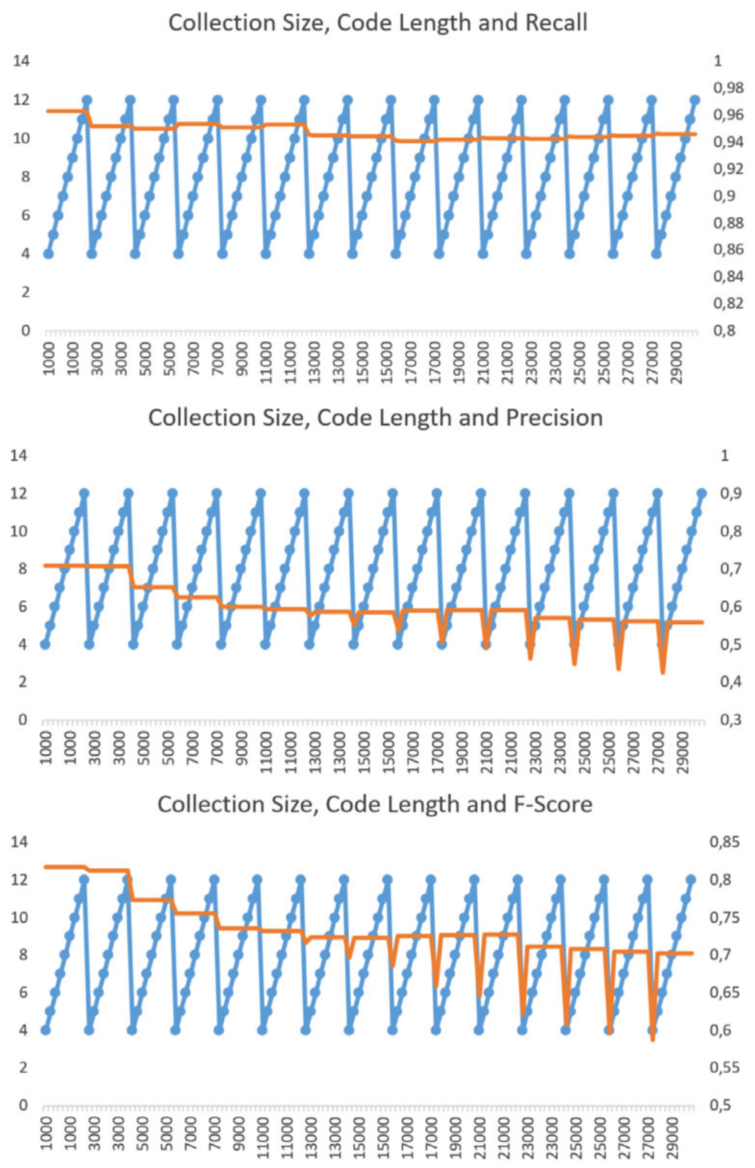


Figure 14. Recall (top), Precision (middle), and F-Score (bottom) as a function of code length (left Y-axis, blue dots) and dataset size (X-axis) of *Soundex*$_{GR}$ in *Dataset H*.

ms to be encoded, while $\texttt{Soundex}_{GR}^{naive}$ in 0.4 s, meaning that it needs 0.016 ms per word. Since $\texttt{Soundex}_{GR}^{comp}$ uses both the implementations to encode a word, it needs 1.016 ms per word.

To compute the $\texttt{Soundex}_{GR}$ codes for each word of the dictionary described in Section 4.7, that is, for more than half a million words, our implementation (using Java 8) takes less than 2 s (specifically 1,684 msecs) using a machine with 1.9 GHz i7, 8MB cache, and 16 GB of RAM.

An implementation of all algorithms, as well as the evaluation datasets, are publicly available at https://github.com/YannisTzitzikas/SoundexGR. Moreover, a tool (editor) for aiding the designer to select the method to be applied is also provided: it shows all codes for the words of the input text, a screenshot is given in Figure 16.

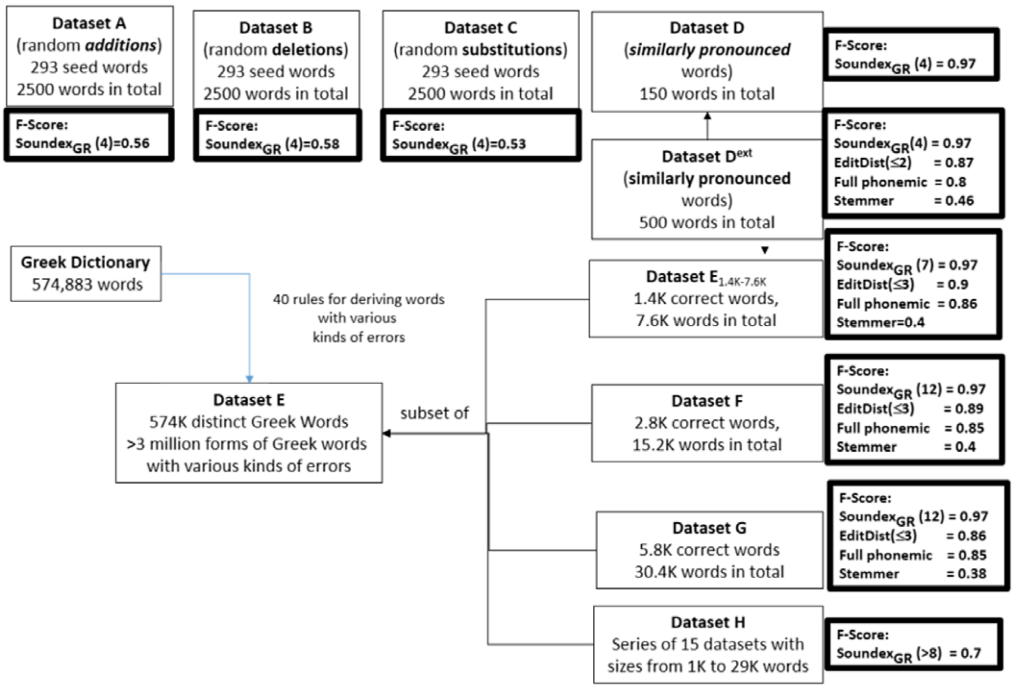
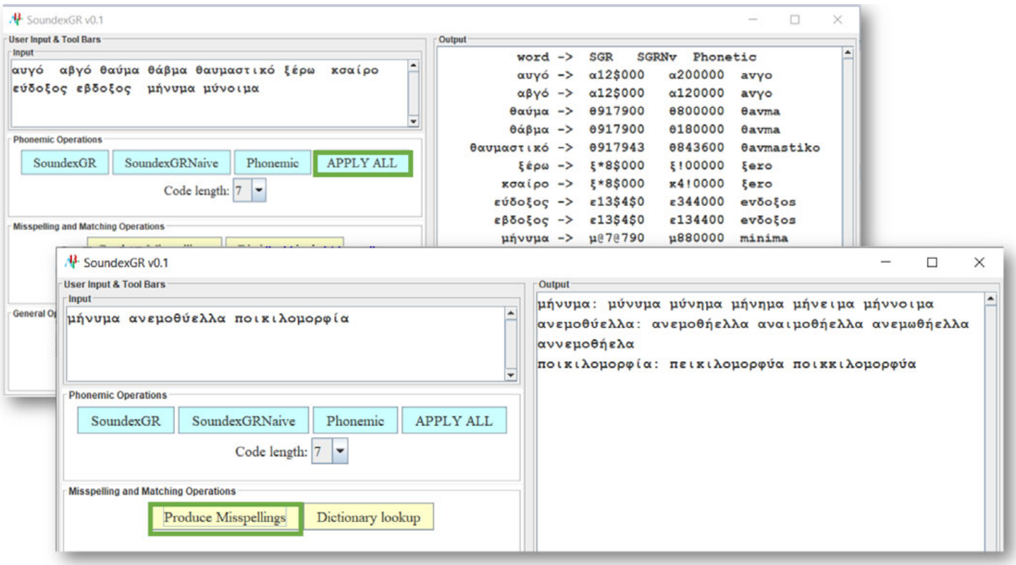

Dataset A
(random *additions*)
293 seed words
2500 words in total
F-Score: Soundex$_{GR}$ (4)=0.56
Dataset B
(random **deletions**)
293 seed words
2500 words in total
F-Score: Soundex$_{GR}$ (4)=0.58
Dataset C
(random **substitutions**)
293 seed words
2500 words in total
F-Score: Soundex$_{GR}$ (4)=0.53
Dataset D
(*similarly pronounced* words)
150 words in total
F-Score: Soundex$_{GR}$ (4) = 0.97
Dataset D^{ext}
(similarly pronounced words)
500 words in total
F-Score: Soundex$_{GR}$(4) = 0.97 EditDist(≤2) = 0.87 Full phonemic = 0.8 Stemmer = 0.46
Greek Dictionary
574,883 words
40 rules for deriving words with various kinds of errors
Dataset E
574K distinct Greek Words
>3 million forms of Greek words with various kinds of errors
subset of
Dataset E$_{1.4K-7.6K}$
1.4K correct words,
7.6K words in total
F-Score: Soundex$_{GR}$ (7) = 0.97 EditDist(≤3) = 0.9 Full phonemic = 0.86 Stemmer=0.4
Dataset F
2.8K correct words,
15.2K words in total
F-Score: Soundex$_{GR}$ (12) = 0.97 EditDist(≤3) = 0.89 Full phonemic = 0.85 Stemmer = 0.4
Dataset G
5.8K correct words
30.4K words in total
F-Score: Soundex$_{GR}$ (12) = 0.97 EditDist(≤3) = 0.86 Full phonemic = 0.85 Stemmer = 0.38
Dataset H
Series of 15 datasets with sizes from 1K to 29K words
F-Score: Soundex$_{GR}$ (>8) = 0.7

Figure 15. A synopsis of the main evaluation results.



```
αυγό  αβγό θαύμα θάβμα θαυμαστικό ξέρω  κσαίρο
εύδοξος εβδοξος  μήνυμα μύνοιμα
```

```
      word ->  SGR     SGRNv    Phonetic
      αυγό ->  α12$000   α200000  avγo
      αβγό ->  α12$000   α120000  avγo
     θαύμα ->  θ917900   θ800000  θavma
     θάβμα ->  θ917900   θ180000  θavma
θαυμαστικό ->  θ917943   θ843600  θavmastiko
      ξέρω ->  ξ*8$000   ξ!00000  ξero
    κσαίρο ->  ξ*8$000   κ4!0000  ξero
   εύδοξος ->  ε13$4$0   ε344000  evδoξos
   εβδοξος ->  ε13$4$0   ε134400  evδoξos
    μήνυμα ->  μ@7@790   μ880000  minima
```

```
μήνυμα ανεμοθύελλα ποικιλομορφία
```

```
μήνυμα: μύνυμα μύνημα μήνημα μήνειμα μήννοιμα
ανεμοθύελλα: ανεμοθήελλα αναιμοθήελλα ανεμωθήελλα
αννεμοθήελα
ποικιλομορφία: πεικιλομορφύα ποικκιλομορφύα
```

Figure 16. A tool for visual inspection of the produced codes, approximate matching, and others.

4.12 Applications

The simplicity and efficiency of the proposed algorithm makes it applicable to a wide range of tasks. It can be exploited whenever we want to find matchings between (written or spoken) μηνυμάτων, μινιμαλισμού, μινιμαλιστικός, μινιμαλιστικά, μινιμαλιστικό, μινιμαλιστικής, μινιμαλιστικέ, μινιμαλιστικές, μηνύματος, μινιμαλισμό, μινιμαλισμός, μηνύματα, μινιμαλιστή, μηνύματά, μινιμαλιστής, μηνύματός, μινιμαλιστική, μινιμαλιστικού, μινιμαλιστικών, μήνυμα, μινιμαλιστικοί, μινιμαλιστικούς, μήνυμά

Figure 17. Suggestions for the mispelled word μοίνειμα based on length code = 6.

descriptions in Greek. In general, these phonetic codes can be used for tackling *Out-Of-Vocabulary (OOV) words*, a problem that occurs frequently and in various contexts. Indeed, the phonetic codes can be exploited for supporting various kinds of matching, depending on the context. As shown in Section 4.10, the way to handle the OOV problem depends on various factors (collection size, kind and percentage of errors, and word lengths). To verify it in a pure matching context, we implemented a prototype matching service where the user enters a word, and the system performs lookup in the dictionary of Greek words (mentioned in Section 4.7 that contains 574,883 distinct words), and if the word is not found, then it suggests to the user a number of approximate matches. Note that this problem is easier in a context where also the frequencies of words are available (e.g., in query autocompletion in web searching); however, we wanted to inspect the behavior of matching if no usage information is available. We implemented the approximate matching by returning all words of the dictionary that have the same $\texttt{Soundex}_{GR}$ code with the word entered by the user. As expected, the returned words depend on the length of the codes that are used. For instance, for the mispelled word μοίνειμα the system, with $\texttt{Soundex}_{GR}$ code length equal to 12, returns two suggestions μήνυμα, μήνυμά. Notice that the edit distance of these words is 4 and 5, making clear the differentiation (and benefit) of this matching in comparison to edit distance-based matching. We obtain the same two suggestions for any code length between 7 and 12.

However, if we further reduce the length to 6, then we get the 23 suggestions shown in Figure 17.

This suggests that the phonetic codes can be used for more sophisticated services as well, for example, if the number of words with the same code is high then we can rank them according to their edit distance. The returned ranked list will include words that sound the same but may have several orthographic mistakes (therefore would be not returned by the edit distance) which will be subsequently ranked with respect to edit distance allowing in this way to control the number of suggestions. An example for the word διάλιμα is shown in Figure 18, demonstrating that ranking with edit distance over the $\texttt{Soundex}_{GR}$ codes gives better results than applying directly edit distance, as the latter includes totally irrelevant words.

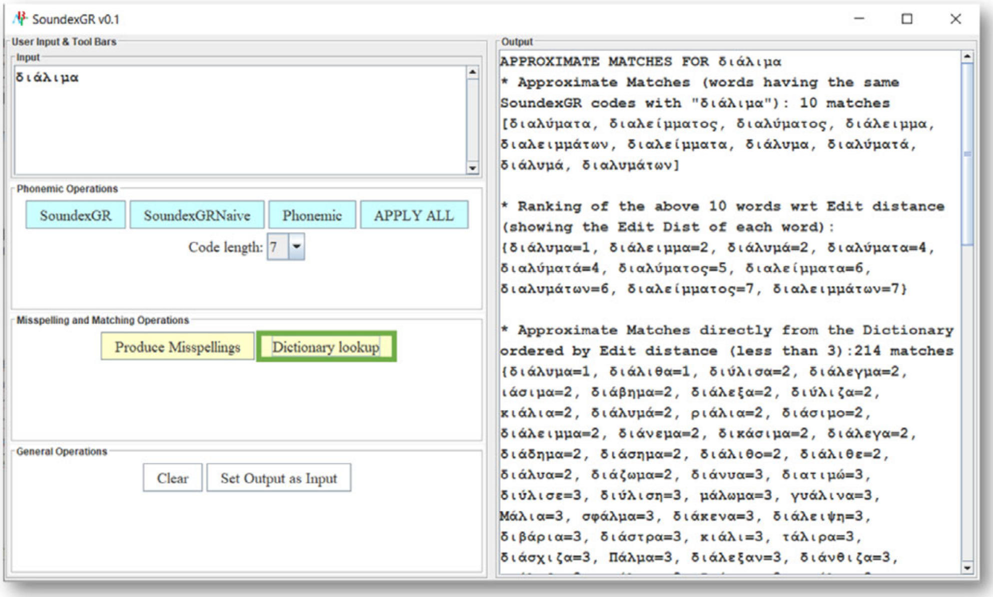
Furthermore, since the codes can be computed once (something that is not possible with the edit distance), this offers a more efficient method for computing approximate matches.

To support the process of designing such services, the application allows testing the above services using various code lengths. It also offers a method that takes as input one word and produces various misspellings, enabling the user to easily pick misspellings for checking the approximate matching (as shown in the bottom part of Figure 16).

4.12.1 Indicative Application Contexts

Below, we sketch how these codes can be used for tackling the problem of *Out-Of-Vocabulary (OOV)* words in various contexts.

- **Autocompletion Services.** Each work *w* in the list of possible query completions (corresponding to the frequent queries according to the query logs) can be accompanied by its



```
APPROXIMATE MATCHES FOR διάλιμα
* Approximate Matches (words having the same
SoundexGR codes with "διάλιμα"): 10 matches
[διαλύματα, διαλείμματος, διαλύματος, διάλειμμα,
διαλειμμάτων, διαλείμματα, διάλυμα, διαλύματά,
διάλυμά, διαλυμάτων]

* Ranking of the above 10 words wrt Edit distance
(showing the Edit Dist of each word):
{διάλυμα=1, διάλειμμα=2, διάλυμά=2, διαλύματα=4,
διαλύματά=4, διαλύματος=5, διαλείμματα=6,
διαλυμάτων=6, διαλείμματος=7, διαλειμμάτων=7}

* Approximate Matches directly from the Dictionary
ordered by Edit distance (less than 3):214 matches
{διάλυμα=1, διάλιθα=1, διύλισα=2, διάλεγμα=2,
ιάσιμα=2, διάβημα=2, διάλεξα=2, διύλιζα=2,
κιάλια=2, διάλυμά=2, ριάλια=2, διάσιμο=2,
διάλειμμα=2, διάνεμα=2, δικάσιμα=2, διάλεγα=2,
διάδημα=2, διάσημα=2, διάλιθο=2, διάλιθε=2,
διάλυα=2, διάζωμα=2, διάνυα=3, διατιμώ=3,
διύλισε=3, διύλιση=3, μάλωμα=3, γυάλινα=3,
Μάλια=3, σφάλμα=3, διάκενα=3, διάλειψη=3,
διβάρια=3, διάστρα=3, κιάλι=3, τάλιρα=3,
διάσχιζα=3, Πάλμα=3, διάλεξαν=3, διάνθιζα=3,
```

Figure 18. Demonstrating approximate matching methods.

Soundex$_{GR}$ code. If the user's input contains a word w' that is not in C, instead of searching for words with small edit distance, the words that have the same Soundex$_{GR}$ can be prompted as well. To support letter-based suggestions, a trie data structure (like the one in Fafalios and Tzitzikas (2015)) of Soundex$_{GR}$ codes can be used for parallel traversal as well, that is, for each letter that is typed by the user we traverse both the trie of frequent queries and the trie of the Soundex$_{GR}$ codes of these queries, and eventually we suggest to the user completions based on the contents of both tries.

- **Retrieval Services.** Each work w in the Vocabulary V of an Inverted File can be accompanied by its Soundex$_{GR}$ code. If the user's query contains a word w' that is not in V (for instance, Cucerzan and Brill (2004) reports that misspellings appear in up to 15% of web search queries), instead of searching only for words with small edit distance, the words that have the same Soundex$_{GR}$ can be used as well. Subsequently, the Soundex$_{GR}$ codes of the words can also be exploited for producing the snippets of the hits that will be displayed in the search results. The snippet of a hit is a small excerpt of that document that contains most of the query words that is computed at query time using sequential text search. Consequently, if the locally stored textual contents of the indexed documents are encoded using Soundex$_{GR}$, then that would speed up the sequential search required for selecting the snippet to display. Other modern applications of real-time searching, for example, methods for linking text to a knowledge base of fact-checked claims (as in Maliaroudakis *et al.* (2021)), for aiding the detection of fake news, can also be benefited by phonetic matching.
- **Named Entity Identification.** Modern methods for Named Entity Extraction rely on pure NLP methods and knowledge-based methods (Mountantonakis and Tzitzikas 2020). The extraction of named entities is usually based on lists of entities (e.g., Countries, etc) which comprise the names of the entities (and alternative names, as in Linked Open Data). Such lists can also contain the phonetic codes of these names to speed up matching and to tackle morphological variations. Indeed, the recent survey by Singh *et al.* (2020) shows that the

components of modern Question Answering systems (that heavily rely on entity identification) are very vulnerable to the morphological variations of the words in the questions that refer to entities.

- **Word Embeddings and ML.** As mentioned in Piktus *et al.* (2019), the existing approaches for producing word embeddings cannot provide embeddings for words that have not been observed at training time. For instance, for the English language, Satapathy *et al.* (2017) used the Soundex algorithm to convert out-of-vocabulary to in-vocabulary and analyzed its impact on the sentiment analysis task, while Satapathy *et al.* (2019) proposed a concept-based lexicon that exploits phonetic features to normalize the out-of-vocabulary concepts to in-vocabulary concepts (Huang *et al.* 2020). An analogous direction could be investigated for the Greek language, since there are already proposals for creating embeddings for the Greek language, for example, the ensemble method described in Lioudakis *et al.* (2019), the method for named entity recognition from Greek legislation described in Angelidis *et al.* (2018), while an evaluation of Greek Word Embeddings is described in Outsios *et al.* (2019), that does not include the more recent Greek BERT Koutsikakis *et al.* (2020). Out-Of-Vocabulary (OOV) words need to be tackled in all cases, for instance, the dictionary that we used contains around 500K Greek words, while Greek BERT Koutsikakis *et al.* (2020) contains embeddings for only 35K words.

In general, applications of phonetic encoding algorithms are widely used in modern information technology, both in the original and modified forms, a detailed list is given in Vykhovanets *et al.* (2020).

5. Conclusion

We introduced a family of phonetic algorithms for the Greek Language by adapting the original Soundex to the characteristics of the Greek Language, and widening the rules, as most modern phonetic algorithms have done. In particular, we introduced $\texttt{Soundex}_{GR}$ and a simpler variation called $\texttt{Soundex}_{GR}^{naive}$, both producing codes of four characters. In brief, before a word is encoded, it is preprocessed and this preprocessing includes identification of cases when a vowel sounds as a consonant in Greek, grouping of vowels that make a different sound when paired together, intonation removal, and dismantling digrams to single letters. Moreover, we defined $\texttt{Soundex}_{GR}^{comp}$ that combines the previous two in the matching process.

To identify which rules have a positive impact on the algorithm, in different error scenarios, we comparatively evaluated these algorithms. To this end, we constructed four evaluation datasets: one with similarly sounded Greek words and three more depending on the kind of error that can happen to a word (letter addition, deletion, or substitution), containing 7650 words in total. The algorithms achieve (precision, recall) metrics that range in (0.90–0.96, 0.40-0.98) for $\texttt{Soundex}_{GR}$, (0.69–0.88, 0.34–0.92) for $\texttt{Soundex}_{GR}^{naive}$, and (0.66–0.86, 0.50–0.98) for $\texttt{Soundex}_{GR}^{comp}$. To synopsize, $\texttt{Soundex}_{GR}^{comp}$ achieves F-Score equal to 0.91 in the dataset with the similar-sounded words. We have also seen that these algorithms behave better (over the evaluation collection) than a Greek stemmer, and we have tested their efficiency over a Greek dictionary comprising more than half a million words. Furthermore, we have seen that the $\texttt{Soundex}_{GR}$ performs much better in comparison to a full phonetic transcription. In an extended dataset that contains common errors, we have seen that $\texttt{Soundex}_{GR}$ achieves the highest F-Score (0.97), outperforming also edit distance-based matching. In bigger datasets (that include long words), $\texttt{Soundex}_{GR}$ preserves its superiority but with code length equal or greater than 6, while the length that gives the optimal F-Score is 12. The effectiveness, the simplicity, and the efficiency of the proposed algorithm makes it applicable to a wide range of tasks. The length of the codes can be configured according to the desired precision–recall performance, and we believe that the experimental results reported in this paper provide help for such configuration; we have seen that the size of the vocabulary, the distribution of word sizes, and the type and percentage of errors determine the code length that gives the optimal performance. Moreover, we have seen that these codes can be used in combination with other methods for approximate matching for achieving more sophisticated matching methods that can be more effective, and even more efficient. The implementation of the algorithm, a stand-alone application for approximate matching that can support the designer on selecting the code length to use, as well as the evaluation datasets, are available at https://github.com/YannisTzitzikas/SoundexGR. Moreover, and to facilitate comparative results, we have created and made public the GMW (Greek Misspelled Words) dataset Tzitzikas (2021), a dataset that contains 574,883 distinct Greek words and 4.32 misspellings per word in average, in total more than 3 million forms of Greek words.

One direction that is worth research is to investigate whether these phonetic codes could be exploited in various deep learning models for NLP for the Greek language (e.g., Lioudakis *et al.* (2019) for word embeddings, Angelidis *et al.* (2018) for named entity recognition from Greek legislation), for making these models more tolerant to misspelled or mispronounced words. Another topic that is worth research is to compute n-grams of such phonetic codes over various corpora and then evaluate whether they can further improve the handling of Out-of-Vocabulary words. Along the same line, since our work is not for word sense disambiguation, for example, the word λόγια in the two phrases "λόγια των ανθρώπων" and "η λόγια παράδοση" will be assigned the same phonemic code even if the meaning is different, N-grams and other more recent methods, either over the original words or over their phonemic transcription, could be investigated in the future for identifying the right sense of a word occurrence.

Acknowledgment. The authors would like to thank Katerina Papantoniou for her feedback and for proof-reading the paper, and the anonymous reviewers for their fruitful comments and suggestions.

References

Ahmed A.F., **Sherif M.A.** and **Ngonga Ngomo A.-C.** (2019). Do your resources sound similar? on the impact of using phonetic similarity in link discovery. In *Proceedings of the 10th International Conference on Knowledge Capture*, pp. 53–60.

Angelidis I., **Chalkidis I.** and **Koubarakis M.** (2018). Named entity recognition, linking and generation for Greek legislation. In *Proceedings of the 31st International Conference on Legal Knowledge and Information Systems (JURIX)*, pp. 1–10.

Arvaniti A. (2007). Greek phonetics: The state of the art. *Journal of Greek Linguistics* **8**(1), 97–208.

Baruah D. and **Mahanta A.K.** (2015). Design and development of soundex for Assamese language. *International Journal of Computer Applications* **117**(9).

Beider A. (2008). Beider-morse phonetic matching: An alternative to soundex with fewer false hits. *Avotaynu: The International Review of Jewish Genealogy* **24**(2), 12.

Christian P. (1998). Soundex-can it be improved? Computers in Genealogy 6, 215–221.

Cucerzan S. and **Brill E.** (2004). Spelling correction as an iterative process that exploits the collective knowledge of web users. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, pp. 293–300.

da Silva M.H.L.F., **da Silva Leite M.T.**, **Sampaio V.**, **Lynn T.**, **Endo P.T.**, **et al.** (2020). Application and analysis of record linkage techniques to integrate Brazilian health databases. In *2020 International Conference on Cyber Situational Awareness, Data Analytics and Assessment (CyberSA)*, pp. 1–2. IEEE.

del Pilar Angeles M., **Espino-Gamez A.** and **Gil-Moncada J.** (2015). Comparison of a Modified Spanish Phonetic, Soundex, and Phonex coding functions during data matching process. In *2015 International Conference on Informatics, Electronics & Vision (ICIEV)*, pp. 1–5. IEEE.

Devlin J., **Chang M.-W.**, **Lee K.** and **Toutanova K.** (2018). BERT: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

Dimitrakis E., **Sgontzos K.**, **Papadakos P.**, **Marketakis Y.**, **Papangelis A.**, **Stylianou Y.** and **Tzitzikas Y.** (2018). On finding the relevant user reviews for advancing conversational faceted search. In Recupero D.R., Dragoni M., Buscaldi D., Alam M. and Cambria E., editors, *Proceedings of 4th Workshop on Sentic Computing, Sentiment Analysis, Opinion Mining, and Emotion Detection (EMSASW 2018) Co-located with the 15th Extended Semantic Web Conference 2018 (ESWC 2018), Heraklion, Greece, June 4, 2018*, volume 2111 of *CEUR Workshop Proceedings*, pp. 22–31. CEUR-WS.org.

Dimitrakis E., **Sgontzos K.** and **Tzitzikas Y.** (2019). A survey on question answering systems over linked data and documents. *Journal of Intelligent Information Systems*.

Elmagarmid A.K., **Ipeirotis P.G.** and **Verykios V.S.** (2006). Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering* **19**(1), 1–16.

Epitropakis G., **Yiourgalis N.** and **Kokkinakis G.** (1993). High quality intonation algorithm for the Greek TTS-system. In *ESCA Workshop on Prosody*.

Fafalios P., **Kitsos I.** and **Tzitzikas Y.** (2012). Scalable, flexible and generic instant overview search. In *Proceedings of the 21st International Conference on World Wide Web*, pp. 333–336. ACM.

Fafalios P. and **Tzitzikas Y.** (2015). Type-ahead exploratory search through typo and word order tolerant autocompletion. *Journal of Web Engineering* **14**, 80–116.

Fourakis M., **Botinis A.** and **Katsaiti M.** (1999). Acoustic characteristics of Greek vowels. *Phonetica* **56**(1–2), 28–43.

Gautam V., **Pipal A.** and **Arora M.** (2019). Soundex algorithm revisited for Indian language. In *International Conference on Innovative Computing and Communications*, pp. 47–55. Springer.

Hood D. (2002). Caverphone: Phonetic matching algorithm. *Technical Paper CTP060902, University of Otago, New Zealand*.

Huang L., **Zhuang S.** and **Wang K.** (2020). A text normalization method for speech synthesis based on local attention mechanism. *IEEE Access* **8**, 36202–36209.

Karakasidis A. and **Verykios V.S.** (2009). Privacy preserving record linkage using phonetic codes. In *2009 Fourth Balkan Conference in Informatics*, pp. 101–106. IEEE.

Karamaroudis C. and **Markidakis Y.** (2006). Mitos Greek Stemmer. https://github.com/YannisTzitzikas/GreekMitosStemmer. Students of CSD-UOC in the context of the course CS463 Information Retrieval Systems.

Karanikolas N.N. (2019). Machine learning of phonetic transcription rules for Greek. *In AIP Conference Proceedings*, volume **2116**. AIP Publishing LLC.

Karoonboonyanan T., **Sornlertlamvanich V.** and **Meknavin S.** (1997). A Thai Soundex system for spelling correction. In *Proceeding of the National Language Processing Pacific Rim Symposium*, pp. 633–636.

Kaur J., **Singh A.** and **Kadyan V.** (2020). Automatic speech recognition system for tonal languages: State-of-the-art survey. *Archives of Computational Methods in Engineering*, pp. 1–30.

Koneru K., **Pulla V.S.V.** and **Varol C.** (2016). Performance evaluation of phonetic matching algorithms on English words and street names. In *Proceedings of the 5th International Conference on Data Management Technologies and Applications*, pp. 57–64. SCITEPRESS-Science and Technology Publications, Lda.

Koutsikakis J., **Chalkidis I.**, **Malakasiotis P.** and **Androutsopoulos I.** (2020). GREEK-BERT: The greeks visiting sesame street. *11th Hellenic Conference on Artificial Intelligence*.

Kukich K. (1992). Techniques for automatically correcting words in text. *Acm Computing Surveys (CSUR)* **24**(4), 377–439.

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

Li D. and **Peng D.** (2011). Spelling correction for Chinese language based on pinyin-soundex algorithm. In *2011 International Conference on Internet Technology and Applications*, pp. 1–3. IEEE.

Lioudakis M., **Outsios S.** and **Vazirgiannis M.** (2019). An ensemble method for producing word representations for the Greek language. arXiv preprint arXiv:1912.04965.

Maliaroudakis E., **Boland K.**, **Dietze S.**, **Todorov K.**, **Tzitzikas Y.** and **Fafalios P.** (2021). ClaimLinker: Linking text to a knowledge graph of fact-checked claims. In *Companion Proceedings of the Web Conference 2021 (WWW 2021)*. ACM.

Medhat D., **Hassan A.** and **Salama C.** (2015). A hybrid cross-language name matching technique using novel modified Levenshtein Distance. In *2015 Tenth International Conference on Computer Engineering & Systems (ICCES)*, pp. 204–209. IEEE.

Mikolov T., **Sutskever I.**, **Chen K.**, **Corrado G.S.** and **Dean J.** (2013). Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pp. 3111–3119.

Mountantonakis M. and **Tzitzikas Y.** (2019). Large scale semantic integration of linked data: A survey. *ACM Computing Surveys (CSUR)* **52**(5).

Mountantonakis M. and **Tzitzikas Y.** (2020). LODsyndesisIE: entity extraction from text and enrichment using hundreds of linked datasets. In *European Semantic Web Conference*. Springer, pp. 168–174.

Newton B. (1972). *The Generative Interpretation of Dialect: A Study of Modern Greek Phonology*, volume **8**. CUP Archive.

Nguyen P.H., **Ngo T.D.**, **Phan D.A.**, **Dinh T.P.** and **Huynh T.Q.** (2008). Vietnamese spelling detection and correction using Bi-gram, Minimum Edit Distance, SoundEx algorithms with some additional heuristics. In *2008 IEEE International Conference on Research, Innovation and Vision for the Future in Computing and Communication Technologies*. IEEE, pp. 96–102.

Ousidhoum N.D. and **Bensaou N.** (2013). Towards the refinement of the Arabic soundex. In *International Conference on Application of Natural Language to Information Systems*. Springer, pp. 309–314.

Outsios S., **Karatsalos C.**, **Skianis K.** and **Vazirgiannis M.** (2019). Evaluation of Greek Word Embeddings. arXiv preprint arXiv:1904.04032.

Papadakos P., **Vasiliadis G.**, **Theoharis Y.**, **Armenatzoglou N.**, **Kopidaki S.**, **Marketakis Y.**, **Daskalakis M.**, **Karamaroudis K.**, **Linardakis G.**, **Makrydakis G.**, **et al.** (2008). The anatomy of mitos web search engine. arXiv preprint arXiv:0803.2220.

Papantoniou K. and **Tzitzikas Y.** (2020). NLP for the Greek Language: A Brief Survey. In *11th Hellenic Conference on Artificial Intelligence (SETN* 2020).

Pennington J., **Socher R.** and **Manning C.D.** (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543.

Philips L. (1990). Hanging on the metaphone. Computer Language **7**(12 (December)).

Philips L. (2000). The double metaphone search algorithm. *C/C++ Users Journal* **18**(6), 38–43.

Philips L. (2013). Metaphone 3. http://aspell.net/metaphone/.

Piktus A., **Edizel N.B.**, **Bojanowski P.**, **Grave é.**, **Ferreira R.** and **Silvestri F.** (2019). Misspelling oblivious word embeddings. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1*, pp. 3226–3234.

Pinto D., **Vilarino D.**, **Alemán Y.**, **Gómez H.** and **Loya N.** (2012). The soundex phonetic algorithm revisited for sms-based information retrieval. In *II Spanish Conference on Information Retrieval CERI*.

Raghavan H. and **Allan J.** (2004). Using soundex codes for indexing names in ASR documents. In *Proceedings of the Workshop on Interdisciplinary Approaches to Speech Indexing and Retrieval at HLT-NAACL 2004*, pp. 22–27. Association for Computational Linguistics.

Russell R. (1918). United States patent 1,261,167. Washington, United States Patent Office.

Russell R. (1922). United States patent 1,435,663. Washington, United States Patent Office.

Satapathy R., **Guerreiro C.**, **Chaturvedi I.** and **Cambria E.** (2017). Phonetic-based microtext normalization for twitter sentiment analysis. In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, pp. 407–413. IEEE.

Satapathy R., **Singh A.** and **Cambria E.** (2019). Phonsenticnet: A cognitive approach to microtext normalization for concept-level sentiment analysis. In *International Conference on Computational Data and Social Networks*, pp. 177–188. Springer.

Sfakianaki A. (2002). Acoustic characteristics of Greek vowels produced by adults and children. *Selected Papers on Theoretical and Applied Linguistics* **14**, 383–394.

Shah R. (2014). Improvement of Soundex algorithm for Indian language based on phonetic matching. *International Journal of Computer Science, Engineering and Applications* **4**(3), 31.

Shedeed H.A. and **Abdel H.** (2011). A new intelligent methodology for computer based assessment of short answer question based on a new enhanced Soundex phonetic algorithm for Arabic language. *International Journal of Computer Applications* **34**(10), 40–47.

Singh K., **Lytra I.**, **Radhakrishna A.S.**, **Shekarpour S.**, **Vidal M.-E.** and **Lehmann J.** (2020). No one is perfect: Analysing the performance of question answering components over the dbpedia knowledge graph. *Journal of Web Semantics* **65**.

Themistocleous C. (2011). Computational Greek Phonology: IPAGreek. In *Proceedings of 10th International Conference of Greek Linguistics*.

Themistocleous C. (2017). IPAGreek: Computational Greek Phonology. https://github.com/themistocleous/IPA_Greek.

Themistocleous C. (2019). Dialect classification from a single sonorant sound using deep neural networks. *Frontiers in Communication* **4**, 64.

Trudgill P. (2009). Greek dialect vowel systems, vowel dispersion theory, and sociolinguistic typology. *Journal of Greek Linguistics* **9**(1), 165–182.

Tzitzikas Y. (2021). GMW - Greek Misspelled Words. http://islcatalog.ics.forth.gr/dataset/gmw.

Vykhovanets V., **Du J.** and **Sakulin S.** (2020). An overview of phonetic encoding algorithms. *Automation and Remote Control* **81**(10), 1896–1910.

Yadav V. and **Bethard S.** (2018). A survey on recent advances in named entity recognition from deep learning models. In *Proceedings of the 27th International Conference on Computational Linguistics*, pp. 2145–2158.

Yahia M.E., **Saeed M.E.** and **Salih A.M.** (2006). An intelligent algorithm for Arabic soundex function using intuitionistic fuzzy logic. In *2006 3rd International IEEE Conference Intelligent Systems*, pp. 711–715. IEEE.

Cite this article: Kavros A and Tzitzikas Y (2023). SoundexGR: An algorithm for phonetic matching for the Greek language. *Natural Language Engineering* **29**, 1305–1340. https://doi.org/10.1017/S1351324922000018