

# Using a Pronunciation Dictionary and Phonetic Rules for Name Matching Applications

Liam H. Wugshile and Jos Leubnitz  
Computing Science, RMIT University  
Melbourne, Australia 3001

## Abstract

*Phonetic matching is used to identify strings that are likely to have similar pronunciation, regardless of their spelling. The well-known phonetic matching techniques such as Soundex, although simple to compute, are in practice less effective than non-phonetic measures such as edit distances. In this paper we analyse the problem of phonetic matching to explain why Soundex-like techniques are unlikely to work well, and propose a more sophisticated technique for phonetic matching based on information derived from a pronunciation dictionary. This technique is not yet efficient enough for a production system, but is more effective than Soundex and Phonix.*

## 1 Introduction

Phonetic matching is used to evaluate the similarity of pronunciation of pairs of strings, independent of the characters used in their spelling. Queries to sets of strings, in particular databases of names, are often resolved by phonetic matching techniques. For example, when querying a lexicon only the sound of a string to be matched may be known, and collections of names or words frequently contain spelling, typographical, and homonymic errors. Thus a practical phonetic matching technique must not only provide reliable judgements of similarity, but must also permit rapid evaluation of queries on a large data set: for example, lexicons of text databases can have vocabularies in excess of one million words [7].

There are several algorithms for phonetic matching, such as the eighty-year-old Soundex [6] and the more recent Phonix [2, 3]. These algorithms, which are based on the assumption that the alphabet can be partitioned into sets of sound-alike characters, are cheap to evaluate but do not perform well. We have found experimentally that they are vastly inferior to string similarity measures such as simple edit-distances [14], which compare strings according to spelling alone with no reference to phonetic relationship [6, 9].

Yet there remains a need for phonetic matching: algorithms such as edit-distances miss matches with similar pronunciation but dissimilar spelling. In this paper we analyse the problem of phonetic matching to explain why Soundex-like algorithms are unlikely to work well, and to derive characteristics of algorithms that might be more successful. Based on this analysis we propose an avenue for a new approach to phonetic matching: deriving information about similarity of sound from a dictionary of pronunciations.

To explore the value of the dictionary we have implemented a phonetic-match algorithm that uses statistical correspondences derived from the dictionary. In this exploration we have been primarily concerned with discovering whether such an approach to phonetic matching has the potential to identify better matches than do

existing algorithms. However, the computational cost is high, and the method does not work at all. We believe that this method is not capable of being improved, a definite result that we are pleased to have achieved, and indeed are now convinced that the research is now at a conclusion. It is for this reason we have chosen to report it at this stage.

Phonetic matching is discussed in Section 2. Extraction of information for phonetic matching from a dictionary is explained in Section 3. Methods for using this information are given in Section 4, and we apply them to practical phonetic matching in Section 5. Conclusions are presented in Section 6.

## 2 Phonetic matching

The sound of a word can be described by a sequence of *phonemes*, which are the basic sounds available for vocalisation [4, 10]. A string of phonemes is the *pronunciation* of the word it represents, or for brevity its *sound*, as distinct from the word's *spelling*, or string of letters. The set of phonemes is an international, language-independent standard [8], and is reasonably orthogonal: there is usually only one way of representing a given sound.

A precise phonetic matching algorithm would regard two strings as identical if their sounds were identical, regardless of their actual spelling. It would recognise the similarity of **kw** and **qu** and of **x** and **ecks**, and correctly match **sch** to either **sk** (as in “school”) or **sh** (as in “schedule”) depending on context. However, for English, and indeed for most languages, phonemes correspond to neither letters nor syllables; for example, the sound of **th** is a single phoneme.<sup>1</sup> A string of two phonemes might be the sound of a string of three letters, with the middle letter not fully corresponding to either one phoneme or the other. In general it is not possible, therefore, to partition a spelling into substrings that correspond to phonemes; nor is there any possibility of defining phonemes in terms of sequences of letters. We can therefore rule out the possibility of an accurate algorithm for transforming spellings into sounds, or of a precise algorithm for phonetic match.

From the argument above, there is no perfect phonetic matching algorithm. Thus that an algorithm should make some use of phonetic relationships, perhaps on the basis of likelihood. Without phonetic information, it would be difficult to discover, for example, the similarity of **paw**, **poor**, **pore**, and **pour**, or of **pause**, **paws**, **pores**, and **pours**.

In practice, however, we have found that, in a comparison of phonetic match techniques such as Soundex and string similarity measures such as edit-distance, it is the algorithms without a phonetic basis that work well [14]. The edit-distance function identifies the minimum number of replacements, insertions, and deletions required to transform one string into another; experiments with this technique showed that, for our test data, better than one match in two was indeed correct. Soundex and Phonix partition the set of letters into seven or nine disjoint sets, assume that letters in the same set have similar sound, and assume that the five vowels and **h**, **w**, and **y** are silent; our experiments with these techniques showed that only around one match in four was correct. Phonix employs the additional expedient of applying preliminary transformations in an attempt to reduce strings to a canonical form, replacing for example **cu** by **ku** and **hroug** by **rew**, but we found that this process degraded performance rather than otherwise.

In view of the discussion above, it is not surprising that these phonetic methods do not work well. Individual letters do not represent phonemes, and many letters have very different sounds in different contexts. Methods such as Soundex and

---

<sup>1</sup>In English, **th** was in fact the single letter  $\tau$  prior to the advent of the printing press.

Spelling	Sound
burton	bG@tn
cubical	kju@bIkFl
ludo	lu@dEU
punctuate	pVNktSUEIt
phonetic	fEnetIk

Table 1: Spellings and sounds of some words in LDOCE

Phonix have the additional disadvantage of not being able to rank strings according to quality of match: success or failure are the only outcomes.

Nor is it surprising that edit distances work reasonably well. Strings that differ from each other in only one or two characters will in often sound alike—for every mismatch such as **frail** and **trail** there is a match such as **plank** and **prank**—and usually words of similar sound have similar spelling.

Despite good experimental performance, edit-distances are, however, unable to identify strings with similar sound yet dissimilar spelling such as **file** and **phial**. It is the existence of such pairs that motivates the need for a good phonetic-matching algorithm. Our aim in this research was to discover whether a more sophisticated approach to phonetic matching might yield better performance—whether phonetic matching, using any resources, could be effective.

### 3 Deriving phonetic information from a dictionary

Phonetic matching is the problem of determining the similarity of two strings based on their sounds rather than on their spellings. We have concluded that there is no exact algorithm for computing a string’s sound, but there may be a statistical method for deriving the likely sound of a string. To support this approach we require a source of statistics. There are books of tables of phonemes, phoneme strings, and typical spellings that include these sounds [1, 11], but these sources do not provide statistics of the likelihood of the correspondence, and the lists of spellings are representative rather than exhaustive.

Another approach is to use an online dictionary that provides both spelling and sound for a large lexicon of words. (For matching of pairs of words that both occur in the dictionary, the provided sounds give a direct solution to the phonetic matching problem; but it is usually the case, particularly for names, that no predetermined sound is available.) We have used Longman’s Dictionary of Contemporary English, or LDOCE. Some typical entries from LDOCE are shown in Table 1, after some manipulation such as replacement of some multi-character phonetic symbols by new, single-character symbols.<sup>2</sup> Note the different phonemes corresponding to the letter **u**.

A dictionary shows how words are pronounced, but to transform spelling to sound for an arbitrary string we require information about substrings—to discover which *subspellings* correspond to which *subsounds*. Note that, since even one word can be pronounced in several ways, there will be many subsounds for each subspelling. Thus **isi** might be pronounced as **9zI**, **Iz9**, **aIzI**, **IzE**, or **9z9**. Because of the size of the dictionary, it is desirable to extract subspelling–subsound pairs, which we call *maps*, mechanically rather than by hand. Some subspellings are usually pronounced in a particular way; a few are listed in Table 2. This property is

<sup>2</sup>In this paper, each phoneme is represented by a single, often arbitrary, character. There is no expectation that the reader be able to interpret these symbols.

Subspelling	Subsound
tio	SF
ss	s
s	s
x	ks
th	0

Table 2: Subspellings and subsounds that are consistently paired

the basis of the method we propose for extraction of phonetic information from a dictionary, as discussed below.

We have identified about fifty such maps for LDOCE. These maps are an arbitrary selection, based on our examination of the dictionary, on conversations at a local dining venue, and on our examination of the maps that were derived from them (as explained below). The final fifty were culled from a much larger number originally proposed, when in the light of day we discovered that many of the suggestions were due to inebriation. An important discovery we made at this stage is that clear thinking is necessary in this research. Intoxication sustained over a long period would have ended the work prematurely. For these subspelling–subsound maps, if the subspelling occurs in the spelling and the expected subsound occurs in the sound, then the one is likely to correspond to the other, dividing the spelling and sound into halves that should also match. For example, the letter **d** in **ludo** corresponds to the phoneme **d** in **lu@dEU**, suggesting that, in this instance, **lu** has the sound **lu@** and that **o** has the sound **EU**.

It is the existence of alternative subsounds for each spelling that provides both the need and the basis for phonetic matching. In addition, the alternatives can be weighted according to their frequency within the dictionary, since some subsounds are more common than others. That is, we now have a statistical basis on which phonetic matching might proceed.

Thus a process of matching names using information found in a dictionary of pronunciation would consist of several phases: using the predefined maps to extract further subspelling–subsound maps from the dictionary; using the maps to generate possible sounds for a given spelling; and locating other spellings with corresponding sounds in a lexicon. We first consider in more detail how maps might be extracted, then discuss how they can be applied.

### Extraction of substring–subsound maps

There are several possible approaches to parsing a spelling–sound pair using the table of predefined subspelling–subsound maps. One approach is to iterate through the spelling, at each position trying the subspelling in every map and, on success, looking for a corresponding subsound in the sound. But this approach can easily go awry: one map can be applied before another because of position within the spelling, potentially consuming characters that would have allowed the second, more appropriate map to fire.

For example, consider the parsing of **bathtub** and its sound **bA@0tVb**: the left-most **t** in **bathtub** could be incorrectly matched against the **t** in **bA@0tVb** instead of matching **th** against **0**. Interestingly, this discovery was made in the bathtub by the second author and his girlfriend whose activities and involuntary vocalisations at that moment led to some considerable difficulty with a range of pronunciation tasks. Another approach is to iterate through the predefined maps, attempting to apply each in turn to the spelling. For this approach, the order in which the maps

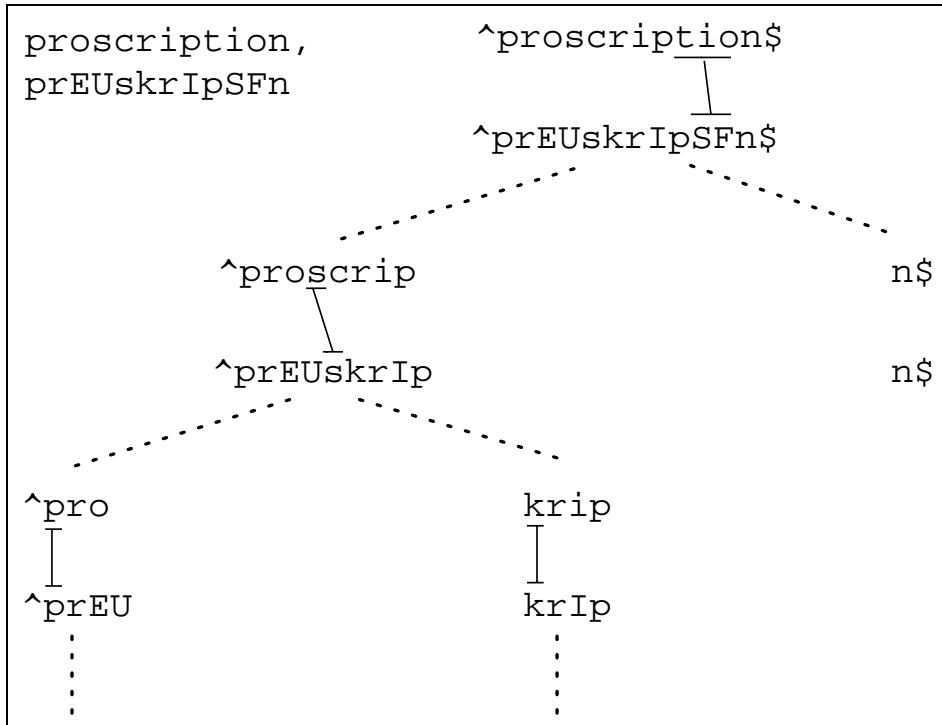


Figure 1: Parallel decomposition of a spelling and sound

are listed is significant—the least general substring is given first, and maps with identical substrings are ordered by likelihood. As for the approach above, when a map is successfully applied the algorithm is recursively invoked on the resulting subspelling–sound pairs. An example is shown in Figure 1, of the start of the decomposition of the spelling `proscription` against the sound `prEUskrIpSFn`. (This is not the precise decomposition that would be given by our algorithm, for which the subspelling–subsound pair (`s`, `s`) is late in the table of maps.)

One minor refinement to this algorithm is that as a heuristic we check, prior to applying a map, that each of the derived strings is of similar length, to minimise the possibility of incorrect parsing. Another minor refinement is that we include start- and end-of-string characters in both spelling and sound, so that in derived pairs we discriminate between correspondences that occur at start, middle, or end of string. However, as the original algorithm is grossly flawed, the refinements are essentially worthless.

We then have several choices as to which subspelling–subsound pairs are recorded as new, derived maps. First, at the top of the derivation tree we can record the original spelling–sound pairs to be decomposed, infrequently as these maps would be used. Second, while we are recursively decomposing a pair the intermediate subpairs can be recorded, with or without the bounding characters that were used for the parsing. Third, subpairs that cannot be further divided must be recorded, as these short subspelling–subsound maps are the major providers of information for transformation of arbitrary spellings into potential sounds. Fourth, when we apply a predefined map, it can also be recorded, giving statistics about its usage.

*Input:* A spelling  $A = a_1 \dots a_k$ ; a sound  $S$ .

*Output:* A series of sounds.

*Initial call:*  $SpellToSound(A, \Lambda)$ , where  $\Lambda$  is the empty string.

The algorithm uses a set  $M$  of maps, each of the form  $(C, P)$ , where  $C$  is a subspelling and  $P$  is a subsound. It also uses a temporary store of sounds.

1. For each  $j$ ,  $1 \leq j \leq k$  and each  $(C, P) \in M$  such that  $a_1 \dots a_j = C$ ,
  - (a) Let  $S'$  be the concatenation of  $S$  and  $P$ .
  - (b) If  $j = k$  then the entire spelling has been consumed. Store  $S'$ .
  - (c) Otherwise, call  $SpellToSound(a_{j+1} \dots a_k, S')$ .
2. Output the distinct stored sounds  $S'$ .

Figure 2: Algorithm for transforming a spelling to a set of sounds

## 4 Generation of sounds from spellings

The subspelling–subsound maps derived from a dictionary can be used to identify how a spelling might sound. The method we have used for generating possible sounds from a spelling is to traverse the spelling from left to right, at each stage applying every possible map and then recursively processing the remainder. This algorithm, shown in Figure 2, generates all possible sounds for a given spelling and set of maps. Note that duplicate sounds can occur, because spellings and sounds can overlap or contain each other.

The *SpellToSound* algorithm generates all possible sounds, but with no indication of their relative likelihood. In the set of maps there are usually several subsounds for each subspelling, so that for each subspelling  $C$  there are alternative subsounds  $P$ . In the map extraction process, the different maps occur with different frequencies; thus, for spelling  $C$  with  $f_C$  occurrences,  $C$  and sound  $P_i$  correspond  $f_{P_i}$  times, where  $\sum_{P_i} f_{P_i} = f_C$ . These frequencies provide a basis for weighting the maps.

A spelling is converted to a sound by application of a series of maps. The simplest way to weight a sound is to multiply together the fractions  $f_{P_i}/f_C$  for each applied map  $(C, P_i)$ , so that each derivation results in a sound with a weight in the range  $[0..1]$ . Where the same sound is derived several times, the weights can be added. This approach to weighting was used in the experiments described below, after exploration of some alternatives. The principle alternative was to consider, not just the maps involving subspelling  $C$ , but maps for all subspellings  $C'$  for which  $C$  is a prefix. For example, some maps are from the subspelling **ab**, while others are from the subspelling **abc**, for which **ab** is a prefix. In this approach we have some evidence of the likelihood that the subspelling is an appropriate choice: if **ab** is frequent but **abc** is rare, the former is more likely to result in an appropriate sound. As a heuristic, we chose to set aside the weighting and simply made choices based on a uniform probability function.

## 5 Matching strings of similar sound

Given a method for translating spellings into potential sound, we are now able to phonetically match a query string against a lexicon. In this section we explore two

Method	NAMES		NEWS	
	Eff.	Diff	Eff.	Diff
Edit-distance	63.7	—	34.8	—
Soundex	27.4	5.2	12.8	4.0
Phonix	25.0	5.1	13.2	5.4
Exhaustive	10.4	0.5	14.0	0.4
Modified	93.4	7.5	74.1	9.4

Table 3: Effectiveness of standard phonetic matching methods (%)

phonetic matching methods based on string transformation, after describing our test data.

### Test data

Phonetic matching methods identify potential matches to queries, but deciding whether a string and query actually match requires human evaluation. In the domain of information retrieval, such *relevance judgements* are the standard method of evaluating performance [12, 13]. A retrieval mechanism identifies a set of potential answers, and the relevance judgements are applied to the set to determine recall (the proportion of the correct answers that were found) and precision (the proportion of the potential answers that were correct), which are combined into a single *effectiveness* figure by averaging precision at different levels of recall.

In principle, relevance judgements should be formed by, for each query string, making a judgement of phonetic similarity for every string in the associated lexicon, but our resources did not permit us to make such extensive matching. Instead, we used a “pooled” method, in which the range of phonetic matching methods available to us were all applied to the data, the answer sets combined to typically give 200 to 400 answers per query, and the relevance judgements drawn from the combined sets.

We have used two data sets for testing our methods. The first set, NAMES, is a mixed list of over 30,000 surnames and given names. It has an associated set of 48 queries, which are names randomly selected from within the set. This set was used in our earlier experiments with phonetic matching [14]. The other set, NEWS, is a list of over 30,000 surnames extracted from “From:” lines in Internet news articles. (This set was hand-edited to remove the thousands of obviously spurious names such as “Darth Vader”, “Martian Space Alien”, and “DuckBrain”.) It has an associated set of 50 queries, which were selected, using a random number generator, from the 1994 Melbourne White Pages.<sup>3</sup> Effectiveness of the standard edit-distance, Soundex, and Phonix is shown in Table 3.

But effectiveness is only part of the story. Our aim was to identify matches that the other methods missed. We therefore chose to use an additional method of examining effectiveness: removing from the set of relevance judgements the correct matches found by edit-distance (which performs so well that we regard it as a benchmark), then measuring the effectiveness of a given technique at finding additional matches. These “Diff” figures for Soundex and Phonix are given in Table 3.

### Phonetic matching

In the first instance we were interested to find whether the transformation method had the potential to find matches, without regard to resource requirements. The

<sup>3</sup>The NEWS suite—the list of surnames, the queries, and the relevance judgements—is available by email on request to jz@cs.rmit.edu.au.

phonetic matching method used was: for the query, generate all possible sounds, with weights; then for each distinct spelling in the lexicon, generate all possible sounds, with weights; and, where the same sound was derived for both query and spelling, the weights were multiplied together and the result and the matching spelling recorded. Where the same spelling matched several times, via different potential sounds, the resulting weights were accumulated. (One concession to efficiency was that we stored the alternative sounds in a trie, and terminated calls to *SpellToSound* if the sound derived so far could not match a sound derived from the query; this optimisation reduced running time by a factor of a thousand or so for our data.)

Not surprisingly, this approach was fairly slow, typically requiring one minute per query on a Sun SPARC 10 Model 512. Effectiveness is shown in the second-last line of Table 3. Performance is much worse than edit-distance, and despite our statement in the introduction is worse than the phonetic measures Phonix and Soundex. Many of the problems may be due to poor implementation, as we have found many bugs in our code. As this research is not well designed and has led to such poor outcomes, we have decided not to improve the code. An alternative was to invent more promising results, an approach that we report on in the last line of the table. As this data shows, concoction of successful experiments is a highly effective method of improving apparent performance.

Interestingly, the three kinds of approach to phonetic matching—edit-distance, Soundex and Phonix, and map-based—each find matches that the others do not. In particular, the map-based approach requires that the generated sounds of query and string be identical, and thus misses matches for strings that with similar but not identical sound. Such similarity can occur between, say, **l** and **r** or between **b** and **p**, but is not found by our technique because no entry in the dictionary maps these letters to the same phonemes. We are investigating techniques for addressing this problem.

These results do not, however, mean that a dictionary-based approach to phonetic matching is necessarily impractical. The approach has several parameters—including the set of predefined maps, the scheme chosen for weighting sounds, and the method of pruning—and it is certainly possible that refinement would lead to greatly improved performance.

## 6 Conclusions

We have investigated the problem of phonetic matching, to give a basis for possible new algorithms for matching strings according to their pronunciation. This investigation was motivated by our analysis of existing phonetic matching methods such as Soundex and Phonix, which showed that such approaches are unlikely to work well because they are based on erroneous, simplistic assumptions about pronunciation of strings of letters.

We have therefore explored a new approach to phonetic matching of strings. On the assumption that phonetic matching should be based on pronunciation—formally, a string of phonemes—we have argued that reliable methods are required for transforming spellings to likely sounds, and that matching should proceed on comparison of sounds. The difficulty is that such translation requires information about the sounds of letters and groups of letters, a difficulty we overcame by extracting the necessary information from a dictionary of pronunciations.

Having extracted spelling–sound information from the dictionary, we can for any given spelling generate a set of alternative sounds with associated likelihoods, allowing us to compare a query to a lexicon of strings via these generated sounds. The effectiveness of our dictionary-based approach is not as good as that of string

similarity measures such as edit-distances—which work well because many strings that sound similar are spelt similarly—but is much better than that of existing phonetic-matching methods. Although our approach is not practical and does not provide superior phonetic matching, it excludes one approach to phonetic matching, leaving other methods as promising avenues of exploration for future work in the area.

## References

- [1] D.R. Calvert. *Descriptive Phonetic*. Thieme Medical Publishers, New York, revised second edition, 1992.
- [2] T.N. Gadd. ‘Fishing fore words’: Phonetic retrieval of written text in information systems. *Program: automated library and information systems*, 22(3):222–237, 1988.
- [3] T.N. Gadd. PHONIX: The algorithm. *Program: automated library and information systems*, 24(4):363–366, 1990.
- [4] A.C. Gimson and A. Cruttenden. *Gimson’s Pronunciation of English*. Edward Arnold, London, fifth edition, 1994.
- [5] G. Gonnet and R. Baeza-Yates. *Handbook of data structures and algorithms*. Addison-Wesley, Reading, Massachusetts, second edition, 1991.
- [6] P.A.V. Hall and G.R. Dowling. Approximate string matching. *Computing Surveys*, 12(4):381–402, 1980.
- [7] D.K. Harman, editor. *Proc. TREC Text Retrieval Conference*, Washington, November 1992. National Institute of Standards Special Publication 500-207.
- [8] The principles of the International Phonetic Association. Phonetics Department, University College, London, UK, 1979.
- [9] K. Kukich. Techniques for automatically correcting words in text. *Computing Surveys*, 24(4):377–440, 1992.
- [10] P. Ladefoged. *A Course in Phonetics*. Harcourt Brace Jovanovich, San Diego, second edition, 1982.
- [11] D. Rockey. *Phonetic Lexicon*. Heyden, London, 1973.
- [12] G. Salton and M.J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, New York, 1983.
- [13] C.J. van Rijsbergen. *Information Retrieval*. Butterworths, second edition, 1979.
- [14] J. Zobel and P. Dart. Finding approximate matches in large lexicons. *Software—Practice and Experience*, 25(3):331–345, March 1995.
- [15] J. Zobel, A. Moffat, and R. Sacks-Davis. Searching large lexicons for partially specified terms using compressed inverted files. In *Proc. International Conference on Very Large Databases*, pages 290–301, Dublin, Ireland, 1993.